UNIVERSITY OF OSLO
Department of Informatics

Autonomic Power Management using a Resource Closure Model

Master thesis

Bengt Olav Olsen

Network and System Administration

Oslo University College

May 26, 2010

Autonomic Power Management using a Resource Closure Model

Bengt Olav Olsen

Network and System Administration
Oslo University College

May 26, 2010

Abstract

The resource closure operator is a new concept within autonomic computing that differs from other approaches in this field in that it is not based on prediction. Earlier models have an assumption that its knowldege about the system is comprehensive enough to be able to predict future behaviour. Instead, the resource closure model acknowledges that it is not possible or even necessary to have such knowledge to be able to manage the system well.

This masters thesis will make developments to a model based on the resource closure operator, and implement it in a specific case of autonomic power management. The chosen case is dynamic processor frequency scaling, which is a method for reducing the processor's power consumption. The ultimate hope of this research is to contribute knowledge towards the goal of better utilisation of computing resources, and, eventually, towards the goal of reducing the overall power consumption of computer systems and data centers.

Three additions to the model will be presented to further enhance it and make it suitable for the chosen application. A proof of concept implementation of frequency scaling will be performed to show the feasibility of such an approach. Estimates of potential energy savings indicate that the use of a resource closure model is a viable approach for autonomic power management.

Acknowledgements

I would like to express my most sincere gratitude to the following individuals for their invaluable help and support during the course of this master thesis:

My supervisor Siri Fagernes for helping me along the way with her support and good advice, for being patient with me when I needed that and for being firm with me when I needed that.

Æleen Frisch for sharing from her vast knowledge of System Administration, for listening when I needed to air my thoughts and ideas to someone, and for all the feedback and practical help she provided.

Prof. Alva Couch for having created the original model of which this thesis is built upon, and for providing me with the most vital instrument for conducting the simulations herein.

And, Asbjørn Heid, from whom I have derived some very useful mathematical insights.

I would also like to thank my family and friends, especially my parents and sister, for their unwavering support throughout my life. It has provided me with a good starting point for my academic pursuits.

My very special thanks goes to my wife Shani, for her unending support, love, and for making this busy period of our lives as easy as it could possibly get.

Last, but not least, I would like to thank my fellow students and staff at Oslo University College, for providing a good learning environment, interesting discussions and lasting friendships.

Thank you!

Ås, 26th May 2010
Bengt Olav Olsen

Contents

List of Tables

List of Figures

List of Equations

List of Algorithms

List of Definitions

Chapter 1

Introduction

1.1 Motivation

The IT sector has over the last couple of decades become a major consumer of electrical power. The explosive growth of the Internet alone has triggered an enormous demand for servers and network infrastructure to provide for its many millions of users [16]. In December 2009 the estimated number of Internet users was as much as 1,8 millions, which is about a quarter of the world's population [22].

During the relatively short timespan from 2000 to 2005, servers worldwide as much as doubled their power consumption. In 2005, all servers in the USA (including cooling and auxiliary equipment) accounted for about 1,2% of the country's total power consumption, and resulted in an electricity bill of $2,7 billion [30].

With the growing awareness of the environmental impact of energy production, it has become clear to the IT industry that they have become a major contributor to global warming. This is one of the reasons behind the current focus on "green computing," which is a collective term for efforts done to make IT more environmentally friendly, and among these, reducing power consumption is an important part [43].

Another important motivation for the industry is to gain cost reductions due to a lower electricity bill. Moreover, future prognoses show an emerging energy crisis where energy supplies will decrease and prices will rise [37]. With this in mind, the incentive to reduce electricity consumption will become even greater, not only because of the price, but because there might not be enough energy produced to meet the demand [16].

One promising and very interesting path for reducing server power consumption is through the concept of cloud computing, enabled by the recent improvements

in virtual machine technologies. Cloud computing is, put shortly, a consolidation of applications and computing services onto central remote servers in a data center, that are accessible over the Internet [44]. There are several benefits that can be achieved by doing this, and one of the most important ones is to utilise the available computing resources better, in terms of processing power, disk storage, administration, space, cooling etc. These are also important factors when it comes to power consumption, and a better utilisation will have a positive effect on the total energy consumption of the world's IT services [16].

The load on a data center will vary a lot. To maximise the utilisation of the computing resources, including any power reduction, unutilised resources should either be shut down, or adjusted down to minimum levels when load decreases, and restarted/readjusted upwards when they once again are needed. Because of the complexity involved, the system needs a high degree of self-management to be able to achieve this [23].

Autonomic computing is one of the leading pardigms of self-management in computer systems. It is named so because of the resemblance to the human autonomic nervous system. Autonomic systems use feedback control, which means that the system gets feedback about how it responds to decisions made by the autonomic control mechanism. This information becomes the basis for future control decisions [18].

The effectiveness of the autonomic control system will be vital for the efficiency of the data center as a whole. The traditional problem with management systems using feedback control is that they require very accurate models of system behaviour to function well [17], and this kind of accuracy in a very complex system is hard to achieve [9].

A recent contribution within the area of autonomic computing is the suggested concept of "Resource Closure Operators" presented by Couch et al in 2009 [10, 11]. It is a general method for predictible high-level management of computer systems in a complex and unpredictible environment, and purports to enable near optimum resource management without any detailed model [10].

1.2 Problem statement

The goal of this thesis is to investigate and evaluate the possibility of using the resource closure model for power management in a computer system, and to make further developments to the model in order to be able to make a proof of concept implementation in a specific case of autonomic power management.

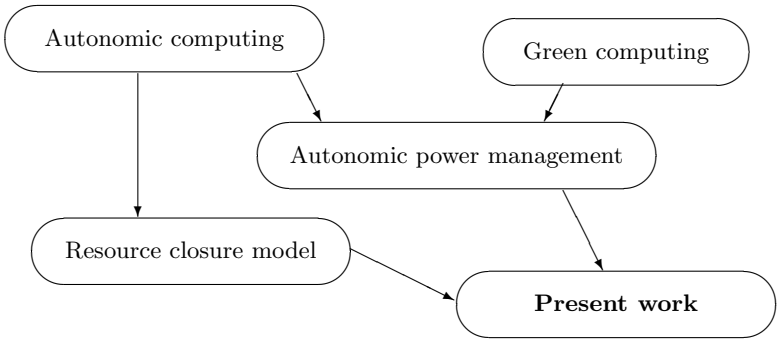

Figure 1.1: Position of this work in the landscape of green computing and autonomic computing.

1.3 Approach

The concept of closures in the context of resource management is currently at a very theoretical level. Research within the topic has mainly been conducted by statistical modelling and simulations [10]. However, a couple of proof of concept implementations have been produced within the related field of configuration management [41, 47].

Further research in resource closure operators will still be dependent on theoretical studies and simulations, as there are many remaining issues to be solved [10]. The work in this thesis will also be theoretical. Simulations will give an indication of whether or not the resource closure model is feasible for this kind of application. It will also identify issues that has to be resolved before the model can be implemented as a proof of concept.

The modelling and simulation part of this thesis has been performed using the *R* environment for statistical computing [45]. The initial source code of the simulation environment and statistical model used in these experiments was kindly provided by Professor Couch [11]. It has been adapted to fit the new model described in this thesis.

A proof of concept implementation has been desirable to better support the results of the simulations and to show the feasibility of the model in a real application. The planned proof of concept will be tightly connected to the theoretical work and simulations, and will be based on the specific case of power management chosen for this thesis.

1.4 Main contributions

The contribution of this master thesis to the research community is two-fold. Mainly, it is a contribution to the research on resource closure operators. The

thesis investigates challenges that formerly have not been addressed in previous research in this field, most importantly the non-linear cost function, priority in value-functions and also the case of limited and discrete resource levels.

In addition, this work is a contribution to the research field of autonomic power management, one of the most important and most challenging fields of research within green computing and autonomic computing today. The scope of this thesis is to evaluate the feasibility of applying a resource closure model in a real setting of autonomic power management.

The ultimate hope of this research is to contribute knowledge towards the goal of better utilisation of computing resources, and, eventually, towards the goal of reducing the overall power consumption of computer systems and data centers.

1.5 Outline

The thesis is organised in the following manner:

Chapter 1: Introduction gives a brief overview of the basic idea behind the thesis, a short explanation of the motivation behind it as well as defining the problem statement.

Chapter 2: Background and literature will give a short, but more thorough introduction to the central topics mentioned in the introduction. It will present the findings from the literature research and give the theoretical basis needed for reading the rest of the thesis.

Chapter 3: Model and methodology presents the model and proof of concept developed during the work with this thesis, and also describes the methodology used in the development process.

Chapter 4: Results will present the results from the simulations and the proof of concept implementation of the new model.

Chapter 5: Discussion and conclusion will summarise and discuss the results of the research and the developed model, as well as present a conclusion.

Chapter 2

Background and literature

This chapter will provide a short introduction to green computing and the issue of power consumption in data centers. Furthermore, there will be an introduction to autonomic computing, including a presentation of the major research in autonomic power management. Finally, there will be a description of the resource closure model used in this thesis, and the central concepts it builds upon.

2.1 Green computing

Green computing has been a hot topic the last few years, as the use of IT systems has grown substantially, and the environmental effects of production, power consumption and waste disposal have become more apparent. One of the main goals of green computing is to use the computing resources as efficiently as possible, while at the same time maintaining or even increasing the overall performance [16].

The discussion around green computing is not completely new; it has been a topic at least since 1992, when the United States Environmental Protection Agency launched Energy Star; a voluntary labeling program designed to identify and promote energy efficient products. Labels are given to the products that meet its energy-efficiency specifications, and its first product group was computers and monitors [16]. It has been estimated that if all computers sold in the US had met the Energy Star requirements, the total cost savings yearly on electricity would be worth around $2 billion, and the reduction of greenhouse gas emissions would be roughly equivalent with the emissions from 2 million cars [46].

Since the early 1990's, the technologies for reducing the power consumption of computers have improved, especially in laptop computers where extending battery time has been an important motivation to achieve better efficiency

[40]. However, the overall power consumption of computers has increased over the same period of time [3]. The focus on development has first and foremost been on processing power [31, 16], and as processors have become exponentially better, following the well-known Moore's law [35], the power consumption has accordingly followed the same curve [15].

Energy, cooling and space in data centers has traditionally been assumed to be both available and affordable, while processing power has been seen as the limiting factor. During the last decade this perception has been turned completely on its head. While processing power has become both very powerful, widely available and affordable, the opposite has happened with cooling, electricity and space [31].

The main focus of green computing today lies on large data centers, with their enormous and sometimes wasteful energy consumption [31]. Enterprise data centers have grown in both number and size, and with an ever increasing need for electricity to run the servers and the necessary cooling, they can easily account for more than half of both the electricity bill and the corporate carbon footprint in the most information-intensive organizations [16].

2.1.1 Adoption and implementation

The biggest challenge today does not necessarily lie in developing new and more energy efficient technology, but rather to adopt and implement the knowledge and technology we already have.

Harmon and Auseklis have identified the main factors driving the adoption of green computing [16]:

The rapid growth of the Internet. This is the main reason behind the increase in data centers. Internet is growing at more than 10% every year, and resource intensive services like music and video downloading, on-line gaming and Voice over IP services are some of the main drivers.

Increasing equipment power density. Servers are getting smaller, but have more processing power. This has increased the power density more than 10 times from 300 watts per square foot in 1996, to more than 4,000 W/ft^2 in 2007.

Increasing cooling requirements. The server power density leads to a higher heat density, which again requires more cooling capacity. Each watt of power used requires another 1-1,5 watts of cooling, and this ratio will increase as the power density increase.

Increasing energy costs. As the energy consumption increases, the electricity costs follow. The expenditure for power and cooling will easily exceed that of the price for the acquisition of the server, and with the estimated increase in energy prices, this will continue to grow.

Restrictions on energy supply and access. As consumption generally rise, price may not be the only restriction, but there might actually not be enough power to meet the demand. Also, the aging infrastructure in some locations may not be able to deliver the electricity needed.

Low server utilisation rates. The utilisation of the servers in large data centers average between 5-10%. Low utilisation of the available resources means that much of the energy and money spent on operating the data center yields no return.

Growing awareness of IT's impact on the environment. Carbon emissions grow proportionally with energy consumption, and as data centers become more energy demanding, the more impact they have on the global environment.

Harmon and Auseklis have also identified strategies for implementing green computing in data centers [16]:

Data center infrastructure. Many data centers are getting old, and most of their equipment is power hungry and inefficient. These will need to be improved or maybe even replaced.

Power and workload management. Using power management software that can dynamically adjust processor power states to match the current workload can save substantial amounts of electricity and money. Estimates from the EPA suggest savings of $25 to $75 per desktop computer per year [46], and even more for servers.

Thermal load management. The increased heat density in data centers demand more efficient and better planned ventilation and cooling systems, and there are several emerging technologies in this area.

Product design. The design of processors is vital for their power usage. For instance, multiple-core processors consume proportionally less power than an equal number of single-core processors. Dynamic power and workload management can also be implemented directly within the processor.

Virtualisation. Data center virtualisation has become the most important way to increase the utilisation of the computing power, which also has a great positive effect on power consumption.

Cloud computing and cloud services. Virtualisation has enabled utility computing where dynamic and high performance computing resources and services have become available over the Internet, where and when they are needed. This way, computing resources can be scaled to meet the demand, which will also give better power management.

2.2 Autonomic computing

As computer systems grow more complex, managing these systems also becomes more and more complex. In a response to this problem, IBM corporation introduced the concept of autonomic computing in a manifesto published in 2001 [18]. In this manifesto, IBM states that the explosive growth of Information Technology infrastructure, processor power and storage capacity can only keep growing for a limited period of time, before it collapses in unmanageability. They have taken inspiration from the human body's self-regulating autonomic nervous system to define what they describe as the next generation of computing, namely autonomic computing.

Since the launch of autonomic computing as a research area, a lot of ambitious research has been started within subjects such as self-configuration, self-protection, self-healing and self-optimisation. However, autonomic computing is still just a grand vision, and will be one of the biggest challenges for the IT industry and academia in the years to come, and it involves many different research fields. IBM researcher J. O. Kephart identifies the main research challenges of autonomic computing in [24].

Even though we are many years away from any fully autonomic computer system, there can be a lot of benefit from improvements made on the way. Kephart and Chess [25] envision a step-by-step path to autonomic systems. The first step is automated functions for collecting and aggregating information to support decisions made by human system administrators. The next step will be advisory functions to offer possible actions for the human operator to choose from. The experience from these functions might enable the autonomic systems to make minor low-level decisions. As the technology progresses and as the human trust in autonomic processes grows, system administrators will make less frequent and predominantly higher-level decisions. Ultimately, self-management will seem natural and unremarkable, and be taken for granted by users and system administrators alike.

Definition 2.1 (Autonomic computing)
Autonomic computing systems are systems that to a certain degree can manage themselves given high-level objectives from administrators [25].

2.2.1 Autonomic power management

The challenges of power management in data centers have motivated research in self-optimising systems for minimising energy consumption. Research has been done within several different areas, and has recently focused on a more holistic approach, taking into consideration such things as memory, I/O and network, in addition to the traditional focus on processor power and cooling [19].

Much research has focused on adjusting processor frequencies according to its current workload. Kandasamy et al [23] propose a very general control mechanism, using a mathematical model that optimises forecasted behaviour based on a limited look-ahead prediction, which can be applied also to other resource management problems. Sharma et al [42] investigate a web server system and propose a mechanism for making the processor run as slowly as possible without violating any quality of service constraints.

Femal and Freeh [14] have investigated non-uniform power allocation and distribution in data centers. They have presented a power allocation method, based on both local and global power limits, that optimises power distribution for server clusters based on a forecasted workload.

Heat management has been the focus of Moore et al [36]. They have examined the impact of factors like air conditioning and physical room layout on the heat and power management of a data center, and presents a prototype method for modeling thermal behaviour.

Khargaria et al [26, 27, 28, 29] have published a series of articles on autonomic power and resource management, developing a more holistic approach to power and heat management. Their experimental results have shown a considerable potential for reducing power consumption using autonomic systems management, while maintaining performance. Their results showed around 72% savings in power with their approach as compared to static power management techniques and 69.8% additional savings with both global and local optimizations [28].

2.3 Resource closure model

The resource closure model is a theoretical and statistical model based on the resource closure operators of Couch et al [10, 11]. It is a contribution to the field of autonomic computing, but differs from other approaches in this field in that it not based on prediction. Earlier approaches are built on an assumption of a model of system behaviour that is comprehensive enough to be able to predict future behaviour.

The resource closure model instead acknowledges that there are influences on the system that are inherently outside of control and not possible to manage; and further that it is not necessary to have complete knowledge of all the influences on a system to achieve sufficient management.

This section will give a brief explanation of the fundamental concepts of the model, as well as a basic technical discussion on how it works.

2.3.1 The closure

A closure is by Couch et al described as a self-managed and predictible component (resource) in a an otherwise open and unpredictible environment. The notion of closures in closed environments existed prior to the work of Couch et al, as many closed-source components are guaranteed to work without any external effects. Their work concentrates instead on how to create and maintain closures in an open environment, where they will be influenced by unknown factors [12].

The term closure comes from programming. It was first described in 1964 by Peter J. Landin [33], and can simplistically be described as an independent environment inside a computer program. Within a closure a variable name is unique, persistent and independent from any other variable with the same name outside the closure. Couch et al apply the term to system administration in a set of 20 principles, each defining properties of a closure in this new context [12].

In the field of configuration management, closures has been implemented and tested in at least two instances, first with an HTTP service closure [41] and later with an IP address closure [47]. Both experiments showed that the closure approach seems to be viable, and that closures do exist for a limited problem domain. These implementations also revealed some of the many challenges that must be overcome before the use of closures in production systems can become a reality.

Definition 2.2 (Closure)
A closure is a structure in which configuration commands or parameter settings have a documented, predictable, and persistent effect on the observable behaviour of the system managed by the closure. A closure in an open system can be described as a "domain of semantic predictability" in an otherwise unpredictible environment [12]. An open system is a system that is influenced by external effects, for instance a computer on a network. A closure in an open system will have a limited domain in where its choices will not have any unintended consequences on the system it manages.

2.3.2 The resource closure operator

The resource closure model is based on a convergent resource closure operator, where each operator is responsible for managing a designated resource. The central concept of the model is the balancing of cost and value associated with a resource to achieve the best possible payoff.

Definition 2.3 (Convergent operator)
An operator is a process that changes the value of a resource variable within the system, by performing an operation on it. A convergent operator has the property that its repeated application will eventually lead to a base state, and no further activity will be registered thereafter. A convergent operator has an idempotent behaviour, which means that its repeated application will always end up with the same result. Through idempotent actions, a convergent operator will converge and bring the system closer to a desired state. Once this state has been reached, a continuous application of the convergent operator will not change the state any further [7, 6].

The resource closure operator is convergent because its repeated application will move the resource closer to its optimal value. Note however, that this optimal value will change over time, so the operator will have to converge towards this moving target [11].

For the operation to be convergent, it requires a method for knowing how to reach the desired state. This is solved using a feedback control loop [11]. Feedback control is a mechanism for sending information about the state of the system or the effects of changes back to the component responsible for making decisions over the system. It is a necessary part of all autonomic systems, and although the technique of feedback control is old, its application in computer systems is fairly new [18].

Definition 2.4 (Feedback control)
"Feedback control is a process by which output or behaviour of a machine or system is used to change its operation in order to constantly reduce the difference between the output and a target value. A simple example is a thermostat that cycles a furnace or air conditioner on and off to maintain a fixed temperature." [18]

2.3.3 The model

Figure 2.1 shows a schematic view of the resource closure model. It shows a system managed by a resource closure, which could be a single web server, a processor or even a cluster of machines.

Functionality

The system consumes a resource R that yields a performance P under load L. The closure Q controls R by either increasing or decreasing it, accordingly. The

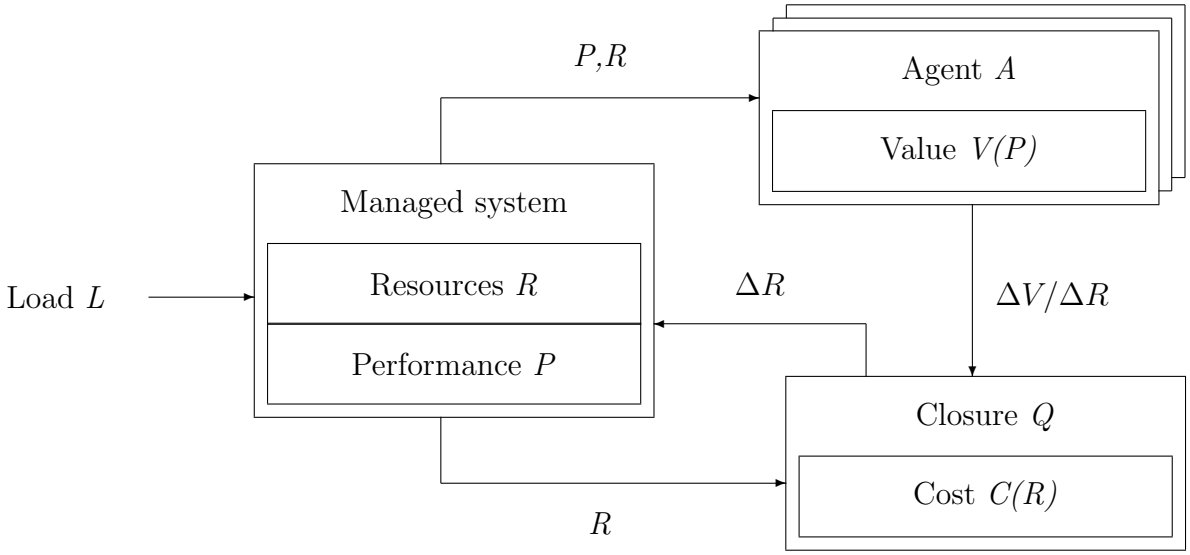

Figure 2.1: A schematic view of the original resource closure model.

load is a representation of all outside forces working on the system, which in an open system is, and will always be, unknown.

The goal of the resource closure is to balance cost and value to achieve the best possible payoff. This is not the same as trying to achieve the best possible performance. Increasing performance is usually just a question of increasing the available resources, and this might well lead to a situation where the cost of the resources surpasses the value they produce.

The value V is evaluated by an external agent A as a function of the system performance. Alternatively, there could be several agents, each with its own set of evaluation criteria. The agents will relay its value estimates to the closure to enable it to make its desicions.

Definition 2.5 (Performance function)
The performance function determines the performance of the system. The performance is defined as the response time of the system, and is a function of resource and load:

$$P(R, L) = L/R \tag{2.1}$$

Definition 2.6 (Value function)
The value-function determines the value of the system's performance. The value decreases as response time increases:

$$V(P) = X - P \tag{2.2}$$

X is a constant of a magnitude that fits with the size of P.

Definition 2.7 (Value estimate change)
The agents provide information about the variation of the value estimate to the closure, as the change in V in relation to the change in R. This is the only information the closure will get about the bahaviour of the system.

$$\Delta V/\Delta R \tag{2.3}$$

In addtion to being informed about the value, the closure also knows the cost C of the available resources in the system. Cost can represent monetary value, which perhaps is the most intuitive, but in principle it could represent anything quantifiable that makes sense in its context.

Using its knowledge about the system, the closure is responsible for balancing cost and value for an optimised payoff π. The closure then calculates a net reward, which the closure use to decide if it will increase or decrease the available resources.

Definition 2.8 (Cost function)
The cost function determines the cost of the available resources. The original model is using a linear cost function where cost equals the available resource:

$$C(R) = R \tag{2.4}$$

Definition 2.9 (Payoff function)
The payoff function determines the total payoff of the system by calculating the difference between cost and the combined value.

$$\pi = V - C \tag{2.5}$$

Definition 2.10 (Net reward)
The net reward N is based on the change in π in relation to the change in R. This value is the basis of the closure's decisions on whether to increase or decrease the available resources. If the difference is positive, Q will give a positive increment ΔR to the managed system, or otherwise, it will give a negative ΔR. The increment/decrement size is a fixed value $|\Delta R|$.

$$N = \Delta\pi/\Delta R \tag{2.6}$$

The closure will sample several net reward values before making a decision. The number of samples is known as the window size, or measurement window. The window size will effect the accuracy of tracking the optimal value.

Limitations

The resource closure operator is a hill climbing algorithm that will converge towards a maximum, which in this case means maximal payoff. It will always converge towards a *local* maximum; however, it is not guaranteed to converge towards the *global* maximum. It is in other words not guaranteed to find the best possible payoff unless the local maximum is also the global maximum. The only way to guaratee this is to make sure that there is only one maximum. This is a problem that has yet to be fully addressed, but which has been assured in this particular model with the convex net value function [11].

Definition 2.11 (Hill climbing algorithm)
A hill climbing algorithm is a mathematical optimisation technique that attempts to maximise or minimise a function of discrete states by finding minima or maxima in a graph representing these states.

The other major limitation of the model is that the algorithm only works in a

highly dynamic environment. It is dependent on change to work correctly and will stop at a non-optimal R if V stays constant [11].

2.3.4 Simulation

The resource closure model has so far only been developed and tested at a theoretical level. Couch et al made a series of simulations of this model [11] using the statistical modelling and computing software R [45]. In addition to showing the feasibility of the model, the simulations uncovered open issues about the model that have to be addressed in the future. It has also identified which variables that must be tuned for the model to perform well.

In the initial phase of this thesis the simulation environment was examined and some simulations were run for the purpose of becoming familiar with the model and the environment. The result of one of these simulations is shown in Figure 2.2.

Environment properties

The simulation environment is an implementation of the model in Figure 2.1, and some additional design choices has been made for the simulation.

In this model, the closure only determines the sign of the increment ΔR. The magnitude of the increment is a constant $|\Delta R|$, because the experiments showed that a varying increment size didn't improve the performance of the model. The increment magnitude did however prove to have a substantial effect on the results, and showed that the size of $|\Delta R|$ will be an important parameter for dynamic tuning.

The load L represents load from outside the managed system, and in the simulation this is represented by a sinusoidal function, which is an easy way of simulating a continuously varying load. The period time is adjustable and will effect the optimal $|\Delta R|$. Using a sinusoidal function ensures that there is only one local maximum for the hill climbing function to converge to.

To be able to judge the effectiveness of the closure, it is compared to the theoretically best value of R, easily calculated as:

$$R_{opt} = \sqrt{L} \tag{2.7}$$

An interesting chicken-and-egg problem appears at startup. The closure operator needs data in order to make any decisions, while the agent needs a change in R in order to make any estimate of value. This problem is fixed simply by making the closure start by incrementing (or decrementing) R by ΔR to create

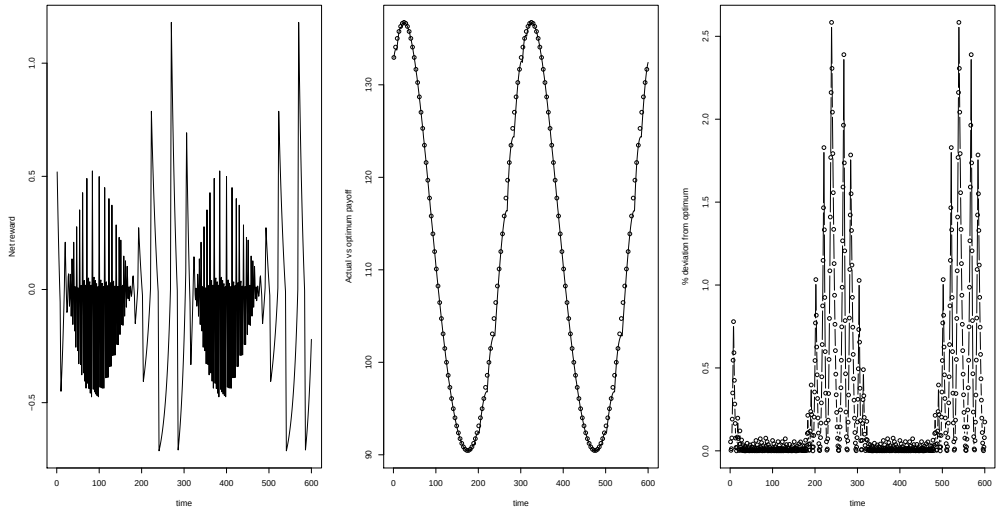

Figure 2.2: Simulation results of the original resource closure model, with net reward, performance in relation to optimum and percentage deviation from optimum. Initial settling time is omitted from the graphs.

experimental data with which the agents make their value estimates. Accordingly, settling time is required before the closure stabilises, so when initiating the simulation, R is set to an initial modest value to reduce the settling time.

Additional parameters include a finite resource bound to limit the amount of resources possible to give, and a window size for the number of observations to base the evaluation on.

Simulation results

The simulation results are shown in figure 2.2. The graph on the left shows the net reward N (see equation 2.6). Its has a seemingly unpredictable behaviour, oscillating wildly between positive and negative values. This behaviour is due to the inaccuracy inherent in the estimator $\Delta V/\Delta R$, because of its lack of knowledge about L [11].

The middle graph is depicting how well the closure performs in relation to the theoretical optimum. The small circles show the optimal payoff, and the line shows how the model estimates the best payoff. It follows the theoretical optimum fairly accurately, even though it gets erroneous input from the agent. The answer to this lies in the fact that the inaccuracy is mitigated by ignoring the magnitude of the estimate, and only considering its sign [11].

The graph on the right depicts the accuracy of the model's behaviour in this simulation, by showing the percentage deviation from the theoretical optimum.

The graph's magnitude is only about 2,5%, and most of the points on the graph lies very close to 0%, indicating that the deviation is small.

Chapter 3

Model and methodology

This chapter describes the model developed during the work of this thesis and also explains the methodology used in the development process.

3.1 Objectives

The main objective of this thesis is to develop and evaluate a resource closure model for use in power management of a computer system, which includes making a proof of concept implementation applied in a specific case of power management, comprising the concepts developed during the theoretical work. This will give an indication of the feasibility of using a resource closure model for power management in a realistic environment.

As described in the background chapter, the resource closure model is a generic model for resource management. There are several steps that have to be taken to successfully implement it as a proof of concept in a specific case.

Step one is to identify which resources in a data center have an influence on power consumption.

Step two is to consider the feasibility of the resource closure model within specific areas of this field, and to choose a specific area to concentrate on.

Step three is to further develop the resource closure model and make necessary adjustments and changes in order to be able to make it work in the chosen case.

Step four is to perform the simulations and analyse the results.

Step five is to finally develop and implement the proof of concept and test it.

3.2 Choice of power management case

The main areas of autonomic power management were identified during the literature research. These areas are presented below, with a short analysis on the expected feasibility for implementing the resource closure model in this area.

Heat management and cooling. Heat management in a data center is undoubtedly a very important area to look at when it comes to saving energy. It is, however, more a topic for thermal management and architecture (room layout) than for system administration. It is worth keeping in mind, though, that a decrease in the amount of power used by the computer equipment will put less strain on the cooling systems, which will give a "double effect" in terms of power reduction.

Power allocation. Better allocation of power resources within a data center is important for effective power usage, especially when availability is limited. Although it is an interesting field of study, it does not seem to be an obvious choice of case for implementing the resource closure model, as it is rather concerning physical distribution, and not internal adjustment, of power resources.

Turning off unutilised devices. The most intuitive and effective way of saving energy is to turn off equipment that is not in use. You will probably save more energy if you always turn off the light when you leave a room than if you install an energy saving light bulb and leave it on all the time. It is the same with servers in a data center. However, it is not equally easy to know when a server is not needed; and even if we knew, it would be an impossible managerial task to manually turn thousands of computers on and off. Therefore, an autonomic system is needed for this purpose.

Using the resource closure model for this purpose might be feasible, and would definitely be a very interesting case. The server capacity of a cluster with the same workload should qualify as the resource to be managed. Cost and value functions could be derived from the operation cost and SLAs of the data center.

Processor frequency scaling. The processor accounts for a major part of a computer's power consumption, and by dynamically lowering the frequency level of the processor, one can get a substantial reduction in the amount of power it consumes. Frequency scaling is also an interesting case for a resource closure model implementation. The processor frequency level seems like a particularly good candidate as an adjustable resource. The load on the system should be easily defined and measurable for evaluation purposes, and finding sufficient cost and value functions for a proof of concept system should be achievable. An additional advantage is that such an implementation would only require a single computer with a dynamically scalable processor, and not an entire cluster of servers.

Holistic approaches. To enable the greatest energy reduction in data centers, it is of course necessary to implement several or all of these methods. To continue the light bulb metaphor, if you change to an energy saving lightbulb as well as turn off the light when you leave the room, it will have an even greater effect. In research studies, there has traditionally been much focus on the processor, as it is the largest consumer of power in a typical server [2].

However, other parts of a computer system also consumes substantial amounts of power, such as disk drives, I/O devices, network, memory and power supplies. Several of these resources could also potentially be interesting cases for the resource closure model, but the feasibility of any of these has not been analysed in this master thesis, and may be the subject of future research.

Definition 3.1 (Processor frequency scaling)
Processor frequency scaling is in reality a voltage/frequency scaling. By lowering the operating voltage of the processor, you can get a substantial reduction in the amount of power it consumes. However, lowering the voltage requires a proportional reduction in frequency [38], so changing the frequency level on a processor will also change the voltage. By voltage/frequency scaling, it is possible to obtain a quadratic reduction in power consumption because the energy consumed by a processor is directly proportional to the square of the operating voltage [2].

$$P = CV^2 f \tag{3.1}$$

(Formula for the power dissipation in a processor, where P=power, C=capacitance, V=voltage, and f=frequency [20])

In this analysis, there are two areas of autonomic power management that immediately looks more interesting as a case for resource closure power management than the others: turning off unutilised devices and scaling the frequency level of a processor.

Both cases deal with scaling down the system when the load is less than the system's capacity. Sharma et al discuss how these two methods of energy saving complement each other when used simultaneously. They also establish that turning on and off servers has high overhead and latency, and that such a method should only be applied on a large time-scale [42].

Scaling of processor frequency levels will make sure that servers will also save energy while active. Because the relation between operating voltage and power consumption of the processor is non-linear, energy savings will be maximised

when the load is evenly distributed among the active servers. Given the use of efficient load balancing systems, we can assume that each server has equal load and that the power management problem needs to be solved for only one machine [42].

The method of power management chosen for this masters thesis is processor frequency scaling. As discussed above, it seems to have the characteristics needed to fit with the resource closure model, and it needs little equipment to perform a proof of concept implementation. Most importantly however, is that it will require further development and some interesting alterations to the model. The following issues have not been investigated previously and will need to be addressed in this master thesis before a proof of concept implementation can be attempted:

- The resource closure model is required to handle a non-linear cost function because the power consumption of a processor is a quadratic function of the frequency [2].
- Scaling of frequency levels in processors can only be done in a few discrete steps. Knowing that rapid switching between frequency levels will lead to overhead [42], this is clearly something that needs to be avoided. A mechanism to accomplish this needs to be implemented in the current continuous resource closure model.
- A data center has SLA requirements which might be different for various workloads. The value function will need to take this into consideration.

3.3 Metrics

The resource metric of the resource closure model in the case of processor frequency scaling would be the frequency level of the processor. There are many varieties of processors with dynamic frequency scaling abilities, but they all have a limited number of discrete frequency levels. For instance, a StrongARM 1100 for handheld devices has as many as ten levels [39], while the AMD Athlon series of microprocessors range from eight to only two frequency levels [1].

System load will be defined as processor utilisation in percentage, which is a common metric for processor load. This will give a number between 0 (idle) and 100 (maximum load), which as a bonus is a very easy scale to relate to.

In Couch et al, the performance metric is defined as being the response time of the system [11]. This will be a good performance metric in the case of power scaling also, since the response time will vary proportionally with the processor speed and inverse proportionally with the system load.

3.4 Additions to the model

As mentioned earlier, processor frequency level scaling has some characteristics that makes it necessary to make further development to the resource closure model. These problems will be further explained in this section, as well as present the solutions suggested in this thesis. These solutions will later be implemented in the simulations and the proof of concept.

3.4.1 Non-linear cost function

The experiments in Couch et al used a linear cost function [11] (see equation 2.4), but since the relation between the processor operating voltage and power consumption is quadratic, it needs to be proven that the model is able to handle a non-linear cost function.

Definition 3.2 (Quadratic cost function)
The quadratic cost function is a non-linear cost function to replace the linear cost function used in the original model in cases where the cost of the resource increase quadratically.

$$C(R) = R^2 \tag{3.2}$$

Whether or not the model could handle the quadratic cost function was not something that could be assumed from the original model, and there was some uncertainty about how well this would work, and if it would work at all. For the purpose of establishing this, applying the problem to the statistical model used by Couch et al in their experimental modelling seemed like a suitable approach.

Two functions needed to be changed in this simulation; the new cost function and the function for finding the theoretical optimum of R. Without the latter, it can not be deduced whether the new cost function is working or not. The purpose of the function is to find the value of R when the payoff function reaches its maximum.

We can calculate this based on the payoff function, first by finding its derivative and finally by solving the quadratic equation against zero, which is when the function reaches its maximum.

$$\begin{aligned} \pi &= V(P) - C(R) \\ &= 200 - LR^{-1} - R^2 \end{aligned}$$

$$\pi'(R) = L/R^2 - 2R$$

$$\begin{aligned} L/R^2 - 2R &= 0 \\ R &= \sqrt[3]{L/2} \end{aligned}$$

This gives the new function for the theoretical optimum of R, that can be used in a simulation to see if the model still works with a quadratic cost function.

$$R_{opt} = \sqrt[3]{L/2} \tag{3.3}$$

3.4.2 Value function priority factor

A resource closure with a quadratic cost function will more strongly prefer the lower resource values than a linear cost function will. This makes the highest resource values more difficult to reach, which is a desired behaviour and the goal of the modified resource closure model presented in this thesis. However, different workloads can have different importance depending on the type of workload, and could for instance depend on the terms set by a service level agreement (SLA). In terms of the resource closure model, this means that the value function needs to take into consideration the varying importance of the workload.

A simple solution to increasing and decreaseing the impact of the value function would be to multiply the result of the value-estimation $X - P$ with a number representing the importance of the current workload. In this thesis, this factor is called a priority factor.

The priority factor is denoted with ρ. When this factor is greater than 1, the total value of the workload will increase in relation to the cost. This will in turn influence the calculated payoff and make the higher resource values more easily attainable and increase the performance of the system when this is necessary or desirable.

Moreover, the priority factor will not have the same impact on all values. On a value of zero, the priority factor would not have any impact at all, but the higher the value gets (or lower if we have sub-zero values), the more impact does it have. This coincides well with the intention of the priority factor, as it is on higher loads that the system needs to increase its performance.

The magnitude of the priority factor may be dependent on the specific type of workload. Generally, three priority levels (1, 2 and 3) should be sufficient, but in theory there is no limit to the size of ρ. If there is a need for finer grained priority levels, decimal numbers could be used instead.

Definition 3.3 (Value function with priority factor)
The value function with priority factor is a modified value function to be used when needing a way to differentiate the value of different workloads. The priority factor is denoted ρ, and is multiplied with the original value function. A priority factor of 1 is in effect equal with the original value function.

$$V(P) = \rho(X - P) \tag{3.4}$$

3.4.3 Resource level steps

The resource closure model as described by Couch et al [11] assumes that the resource variable R can be changed on every iteration of the feedback control loop. It is in fact dependent on this change in value, as it is the basis for the decision making algorithm of the closure itself.

Scaling of frequency levels in processors can only be done in a few discrete steps because processors have a limited number of frequency levels, normally ranging between two and ten different levels. This is not a problem with the model as such, as the legal values of R can be discrete, and changes are done in discrete steps of an increment value $|\Delta R|$ [11].

In theory, R could be limited to the available frequency levels, and the increment value could be one frequency level up or down. However, this would mean that it is likely that the frequency level of the processor would change at every iteration. Depending on the time between every iteration of the feedback control loop, this constant switching could lead to an overhead, and possibly even increase the total power consumption rather than decreasing it [42].

There are at least two ways to approach this problem. One way would be to apply a test that checks if sufficient time has passed since the last frequency change before allowing another change. The apparent advantage to this is the simplicity of the concept and that it would be easy to implement. However, this would be an overruling of the decision made by the resource closure operator and could possibly affect the functionality of the feedback loop and the closure operator because the feedback will not be based on the change recommended in the previous iteration, thus making it less optimal.

Another approach is to integrate a solution into the model, by defining two different resource values; one for the discrete actual resource level and another

for the continuous recommended resource level. The closure will only deal with R_{rec}, whereas R_{act} would only change when R_{rec} reaches a pre-defined threshold. By defining different values of R_{rec} for the upwards and downwards thresholds (T_{up} and T_{down}), oscillation between two discrete resource levels is avoided, provided ΔR is sufficiently small.

Definition 3.4 (Resource level hysteresis function)
The resource level hysteresis function is an algorithm for preventing oscillation between two discrete resource levels when the resource closure recommendations lie in the border area between two levels. It defines a span between the upward and downward thresholds, so that when changing discrete resource level, it can not revert back to the same level for a certain number of iterations. In physics, this behaviour is known as hysteresis.

Algorithm 1 Resource level hysteresis function

```
if ΔR > 0 then
  calculate T_up from R_act
  if R_rec > T_up then
    if R_act ≠ R_max then
      increase R_act
    end if
  end if
else if ΔR < 0 then
  calculate T_down from R_act
  if R_rec ≤ T_down then
    if R_act ≠ R_min then
      decrease R_act
    end if
  end if
end if
```

The pseudo code of the solution is presented in algorithm 1 and works as follows:

1. The sign of ΔR is investigated to see if R_{rec} is increasing, decreasing or if it stays the same. If it does not change, i.e. $\Delta R = 0$, then nothing will happen.

2. The threshold value is calculated depending on the direction of the change and the current R_{act}. The threshold is lower on a downwards change (T_{down}) than it is on an upwards change (T_{up}) so that there is a small span of values of R_{rec} that maps to two different values of R_{act}. This ensures that if R_{rec} for a while stays in the area bordering two different frequency levels, it will not cause an oscillation between them, as long as the increment size of ΔR is small enough to require at least a couple

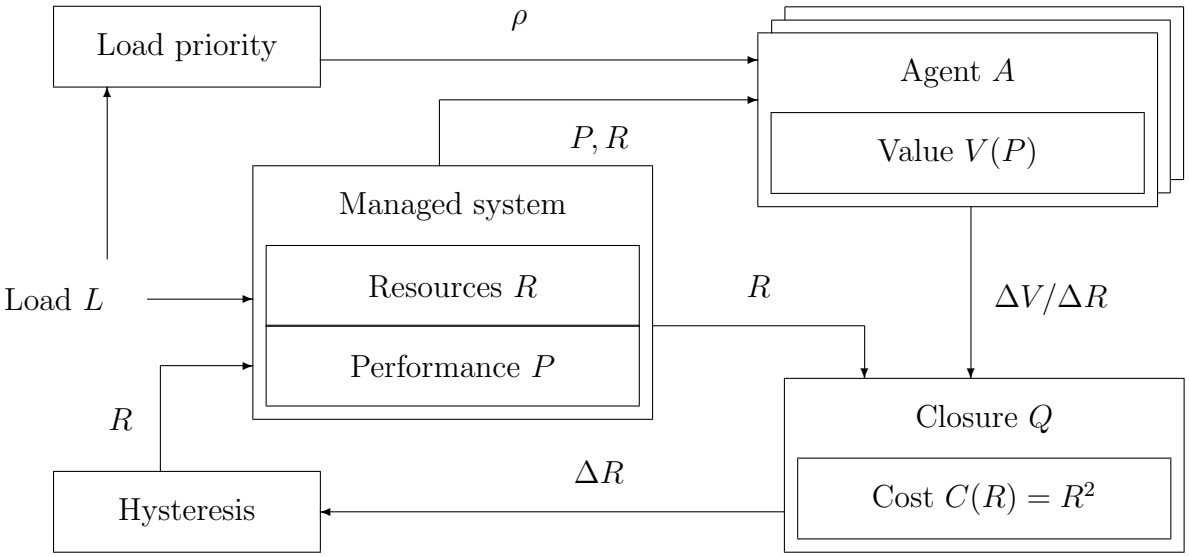

Figure 3.1: The new resource closure model, comprising the additions presented in this thesis.

of iterations of the resource closure before R_{rec} can cross this span of values and reach the opposite threshold. The size of this span and the threshold calculation are design choices to be made in accordance with the magnitude of R.

3. After calculating the threshold, a check is made to see if R_{rec} has reached or passed it. On an upwards change, it needs to pass the threshold, whereas on a downwards change, it only needs to reach it. This is a design choice, and not of vital significance to the algorithm.

4. Before increasing or decreasing the value of R_{act}, it will check if the next resource level exists. If the current resource level is either the minimum or maximum available (R_{min} or R_{max}), no change is made.

3.5 Design of simulations

Simulations were run in the R environment for statistical computing, by applying the relevant changes to the resource closure model's simulation environment.

3.5.1 Original model with non-linear cost function

The first issue that had to be solved by simulation was that of the non-linear cost function. The simulation environment was identical to that described in the background chapter, except for the following changes:

- The cost function was changed to $C(R) = R^2$ (see equation 3.2).
- The function for calculating theoretical optimum was changed to $R_{opt} = \sqrt[3]{L/2}$ (see equation 3.3).
- The value function was changed to $V(P) = 1000 - P$, because the magnitude of the values of P has changed. Raising the constant to 1000 was enough to prevent the value of V to fall beneath zero.

Results of the simulation are shown and explained in the results chapter.

3.5.2 New model

The simulation of the quadratic cost function proved it to be working, and the simulations could be further developed to accommodate for a proof of concept.

Load

The first change that had to be made to the model was to redefine the load. In Couch et al, there was a sinusoidal load function operating between 1000 and 3000 [11]. With the new definition of load as processor utilisation in percent, the load will instead be a function operating between 0 and 100.

Different types of load are used in the simulations:

Constant load. A non-varying load is a good start for observing the behaviour of the model when the load is not changing. The simulations will be performed with low, medium and high load, 1%, 50% and 100% respectively (see figure 3.2).

Sinusoidal load. This is a simple way of generating a varying periodic load, and is the choice of Couch et al for testing their original model [11]. The simulations will be performed with a sinusoidal load varying between 0 and 100 (see figure 3.3).

Realistic load. The varying load used is profiled on an actual system load measured over the course of a few hours, and modified to better fit the purpose of testing. It starts with a long period of relatively high load, before it plummets down to a very low load. It then slowly rises to a

medium high load, before descending again. Again, it rises to medium load, this time ascending and descending more quickly. It ends with a period of low load, with a single extreme spike. These different characteristics will try to illustrate the performance of the model under different circumstances (see figure 3.4).

Noise. For some of the simulations using the different types of load mentioned above, a normally distributed noise will be added to uncover the model's vulnerability to measurement errors. The simulations will use the same noise-generator as used in Couch et al, represented by the Gaussian error term $e(0, \sigma)$, where σ is the standard deviation [11].

Resource parameters

The resource parameter of this model is the frequency level of the processor. The processor used in the proof of concept implementation has five frequency levels, as listed in table 3.1. As discussed earlier, the model operates with two different resource parameters, the actual frequency level R_{act} which is restricted by the discrete frequency levels of the processor, and R_{rec} which is the frequency recommended by the model and has to be mapped to a legal R_{act}.

Although R_{act} is limited to the range between 2 and 0,8 GHz, this is not necessary for R_{rec}, because if it is surpassing the highest or lowest legal resource level for R_{act} it just means that R_{act} stays at its highest or lowest resource level, respectively. So as long as the value of R_{rec} stays within a reasonable range of R_{act}, this is not a problem. It is in fact desirable, especially if the upper or lower threshold lies very close to the highest or lowest legal value of R_{act}, or else you risk that it is practically impossible to reach these frequency levels.

The size of the resource increment $|\Delta R|$ is according to Couch et al [11] a critical parameter, and also a good candidate for dynamic tuning. Finding a good size for the increment is important, but fine-tuning to find "the perfect increment size" is not necessary to get a working proof of concept. As earlier mentioned, it is necessary to have an increment size low enough to prevent an oscillating behaviour of R_{act}, but if it is too small the convergence time will be too slow for the model to react quickly enough to changes.

Finding the increment size to use in the simulations will be done by testing several values in simulations, and choose one that yields good behaviour. A first estimation suggests a $|\Delta R|$ somewhere between 0,01 and 0,1. Some other values must also be chosen for the same criteria; these include window size, period of sinusoidal load and the σ-value of the Gaussian error term. These design choices are explained further in the results chapter.

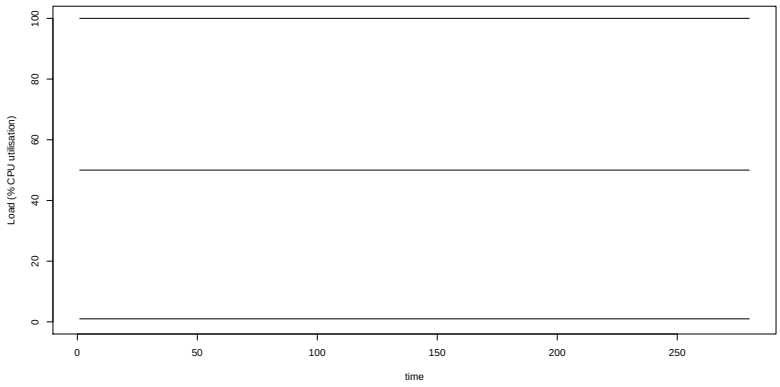

Figure 3.2: Constant input load of 1%, 50% and 100%

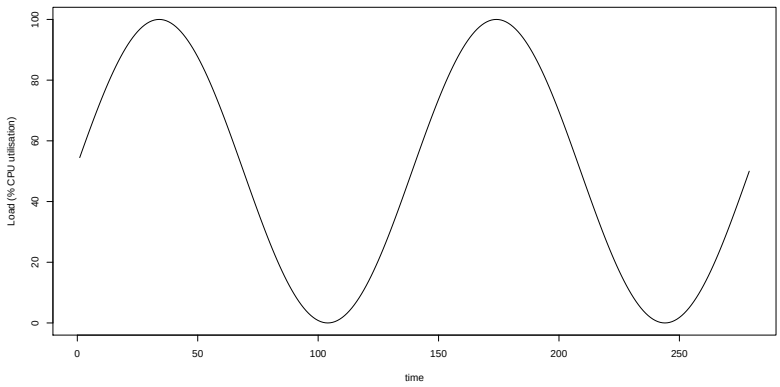

Figure 3.3: Sinusoidal input load

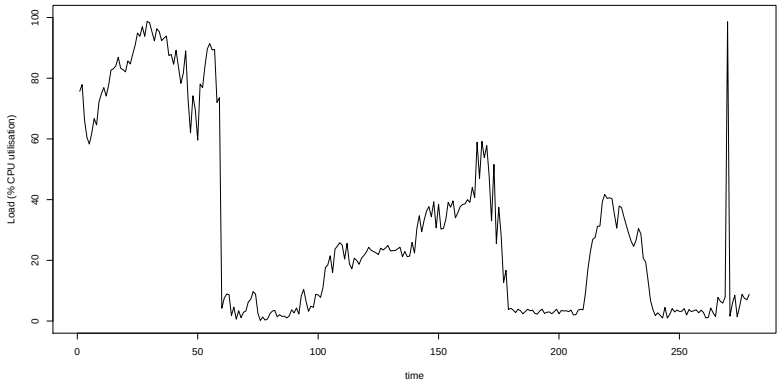

Figure 3.4: Realistic input load

Table 3.1: Processor frequency levels of Intel Pentium M 760, including estimated operating voltage and power dissipation.

Level	Frequency	Voltage	Power dissipation
5	2,00 GHz	1,315 V [1]	27 W [3]
4	1,60 GHz	1,114 V [2]	19.9 W [4]
3	1,33 GHz	0.979 V [2]	15.3 W [4]
2	1,07 GHz	0.849 V [2]	10.8 W [4]
1	0,80 GHz	0,714 V [1]	6.5 W [4]

[1] Voltages for highest and lowest frequency level are found in the processor's data sheet [21].

[2] Voltages for the three levels in between are not noted in the data sheet, and therefore had to be estimated. The estimations are based on an assumption of a linear relation between voltage and frequency, and may vary from the actual voltage levels of the processor.

[3] Maximum power dissipation is noted in the data sheet.

[4] Estimated using the power dissipation formula (equation 3.1), based on maximum power dissipation and estimations of voltage and capacitance levels.

Definition 3.5 (Resource parameters)
This thesis operates with three different resource parameters:

- *The optimal resource R_{opt} is the theoretical optimal resource value calculated from the input load. This resource value is not known by the resource closure model.*
- *The recommended resource R_{rec} is the linear resource value recommended by the closure.*
- *The actual resource R_{act} is the discrete resource value which is the output of the hysteresis function, and is the actual value used for setting the processor frequency.*

Value and cost

Value is dependent on the performance and the priority factor descibed earlier, while cost is dependent only on the resource. Using the formula for calculating the performance (see equation 2.1), we can find the highest attainable performance value, when load is at maximum and resource at minimum: $P = 100/0,8 = 125$. Choosing this number as the constant X in the value function (see equation 2.2) will give a value between 0-125. Adding a priority factor, the new value function will be:

$$V = \rho(125 - P) \tag{3.5}$$

In these simulations, the priority factor ρ will be limited to three levels.

$$\rho \in \{1, 2, 3\} \tag{3.6}$$

Level 1. This is the lowest level, where the lower resource levels will be strongly preferred by the closure. Simulations will show if the higher levels is even attainable at full load.

Level 2. This is the medium level, where higher resource levels will be more easily reached with high load.

Level 3. This is the highest level, where the higher resource levels should be easily reached, even with medium load.

The quadratic cost function (see equation 3.2) will give a maximum value when the resource is at its maximum level (2 GHz): $C = 2^2 = 4$. The range of values between its minimum and maximum is a lot lower than what we have in the value function. Multiplying it with a constant factor of 10 will make the magnitude more sensible:

$$C = 10R^2 \tag{3.7}$$

Changing the value- and cost functions requires a recalculation of the theoretical optimum of R. As earlier, this is calculated from the payoff function, using the modified value and cost functions (equations 3.5 and 3.7). The optimal resource value for the new model is:

$$R_{opt} = \sqrt[3]{\rho L/20} \tag{3.8}$$

3.6 Proof of concept design

The proof of concept makes use of the new model to control an actual processor. It was implemented on a laptop computer with Ubuntu Linux operating system and a 2 GHz Intel Pentium M 760 processor [21] with a five-level dynamic frequency scaling, ranging from 800 MHz to 2 GHz (see table 3.1).

A script for controlling the processor frequency levels was developed in Perl. It uses the results from the realistic load simulations of the resource closure model

in R as its input data. The script calls the linux kernel CPUfreq subsystem [4] to control the frequency settings. The full source code of the Perl script is found in Appendix A.

The script starts by fetching the frequency levels that are available on the system's processor. This information is stored in the file `scaling_available_frequencies` in the CPUfreq directory. It will continue by reading in the file containing the recommended frequency levels from the resource closure model implementation in R. It will use these recommendations as its input for controlling the processor.

The script implements the resource level hysteresis algorithm presented on page 26. The thresholds can be set as a parameter in the script, as a percentage difference from the current frequency level. The thresholds used in the proof of concept implementation are found in table 3.2.

The script iterates through all the recommendations made by the closure implementation. When reaching a threshold, either upwards or downwards, it will first check if the processor is already at the highest or lowest available frequency level. If it is not, it will make a call to the CPUfreq subsystem with the command `cpufreq-set -f <frequency>` [5] to set the new frequency level; or else no action will be performed.

Between each iteration the loop will wait for a predefined time, before fetching the next recommendation. Information about each iteration performed will be logged to file, including the number of the iteration, recommended resource value, next threshold, whether the threshold has been reached and the frequency level of the processor after the iteration has finished. The loop will run until there are no more recommendations from the closure.

Table 3.2: Thresholds in the resource hysteresis function of the proof of concept implementation.

Current freq. level	**Upward threshold**[1]	**Downward threshold**[2]
2.00 GHz	— [3]	1.60 GHz
1.60 GHz	1.76 GHz	1.33 GHz
1.33 GHz	1.46 GHz	1.07 GHz
1.07 GHz	1.18 GHz	0.80 GHz
0.80 GHz	0.88 GHz	— [4]

[1] Upward threshold is defined as 10% above current frequency level.
[2] Downward threshold is defined as equal to the below frequency level.
[3,4] The frequency is already at maximum/minimum level, and can not ascend/descend any further.

Chapter 4

Results

This chapter shows and explains the results that have been achieved during the work with this thesis. The results are divided in two parts, simulation results and the results of the proof of concept experiments.

4.1 Simulations

A series of simulations were made to support the development of a proof of concept implementation. The results of these simulations will be presented here.

4.1.1 Design choices

Some of the design choices had to be made by trying different values in simulations, finally choosing the values that seemed the most fitting and at least sufficient for the proof of concept implementation. These trial-simulations resulted in the following design choices, that will be used in all graphs unless otherwise stated:

The increment size $|\Delta R|$ was set to 0.05, which is a compromise between a low enough increment to avoid oscillation of R_{act} and sufficiently high to react sufficiently to changes. It is not an optimal value for convergence time, as will be shown later.

The window size is the number of samples of value estimate changes that are used to calculate the net reward. A small window size of 3 was chosen, which is smaller than in the original model. If the window size is too large, it will amplify estimation errors [11], and the estimation errors in these simulations are already amplified by the quadratic cost function.

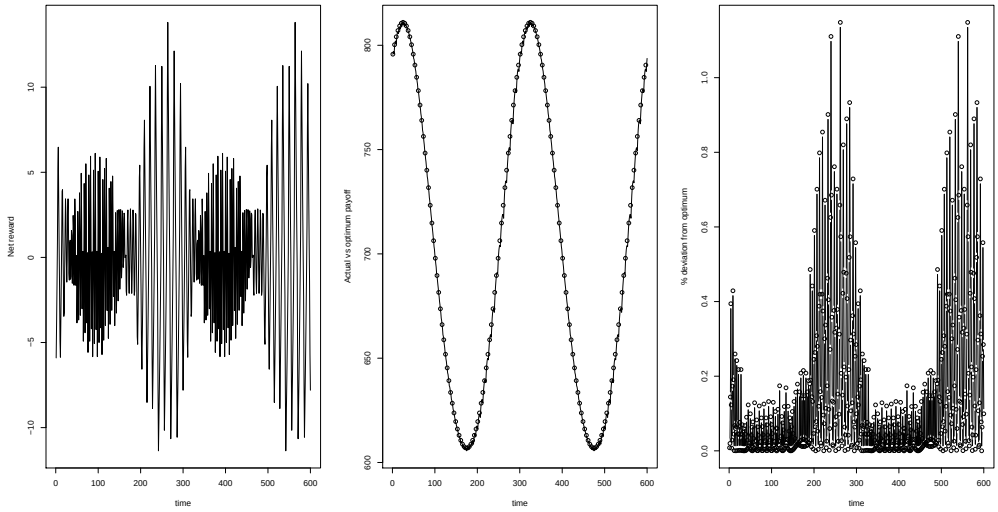

Figure 4.1: Simulation results of the resource closure model with a quadratic cost function. The graphs are directly comparable with the graphs in Figure 2.2 on page 16. Initial settling time is omitted from these graphs.

The period time of the sinusoidal load function was set in accordance with the realistic load, which has a length of 280 measurements. The period time was set to 140, which gives the simulation two full periods of a total length of 280.

The σ-value for noise was set to 1, which is equivalent to 1% since the load is in the range from 0 to 100.

4.1.2 Non-linear cost function

The first simulation was run to find out if the resource closure model was able to handle a non-linear cost function. Except for the changed cost function and directly related changes, this simulation was identical to the simulation of the original model, which is explained in the background chapter.

The results of this simulation are shown in figure 4.1. The graph to the left shows the net reward N (see equation 2.6). It has similar qualities with the original, but it is oscillating more frequently, and with a much higher amplitude. This indicates that the inaccuracy of the estimator $\Delta V/\Delta R$ is amplified by the quadratic function.

However, as we can see from the middle graph, this does not seem to have any negative effect. The payoff function closely follows the theoretical optimum, just as well as it does in the original model. This is due to the nature of the

closure model, because it only considers the sign of the net reward, not its value. The right graph supports the observation, by showing that the deviation from optimum is very low. The highest peaks of the graph are actually less than half of what they are in the original model, although the values seem to be a bit more scattered.

With the success of this simulation, it is concluded that the resource closure model works well with a quadratic cost function, and that it can be developed further.

4.1.3 Simulations with constant load

The simulations with constant load were conducted with three different levels of load; low load (1%), medium load (50%) and full load (100%), each with priority factors of 1, 2 and 3. The results with medium load are shown in the graphs in figure 4.2, and the graphs in Figure 4.3 show the effect of noise. For the equivalent graphs of low and full load, see Figures B.1-B.4 in Appendix B.

The first observation is that at constant load, the payoff function follows the theoretical optimum very closely, with all loads and with all priority factors. It is, however, sub-optimal for a short period in the beginning. This is while the resource closure settles, and is as expected. Settling times have not been omitted from any of the graphs in the simulations that follow.

It is also observed that the graph showing the recommended processor frequency never flattens, even if the payoff graph is nearly flat. This happens because the closure overcompensates at each change of direction, which leads to an oscillation around the theoretical optimum. The oscillation-effect can be reduced by lowering the increment size, but this will also lead to a much slower convergence time, as can be seen in figure 4.4.

Another observation is that the priority factor has very little influence on the recommended resource level on low load (Figure B.1), and much more significant influence on higher loads (Figures 4.2 and B.2). A priority factor of 1 generates the same result as the original model without priority factors would have done.

According to Couch et al, noise, like for instance measurment errors, affects the $\Delta V/\Delta R$ estimate by making it more inaccurate, which again affects convergence time [11]. This creates a substantial oscillation around the theoretical optimum, which we also can observe in these simulations. Moreover, because the resource recommendation is already oscillating, the effect becomes significantly greater in this simulation. An additional observation is that the settling time becomes longer with noise, again an effect of the prolonged convergence time.

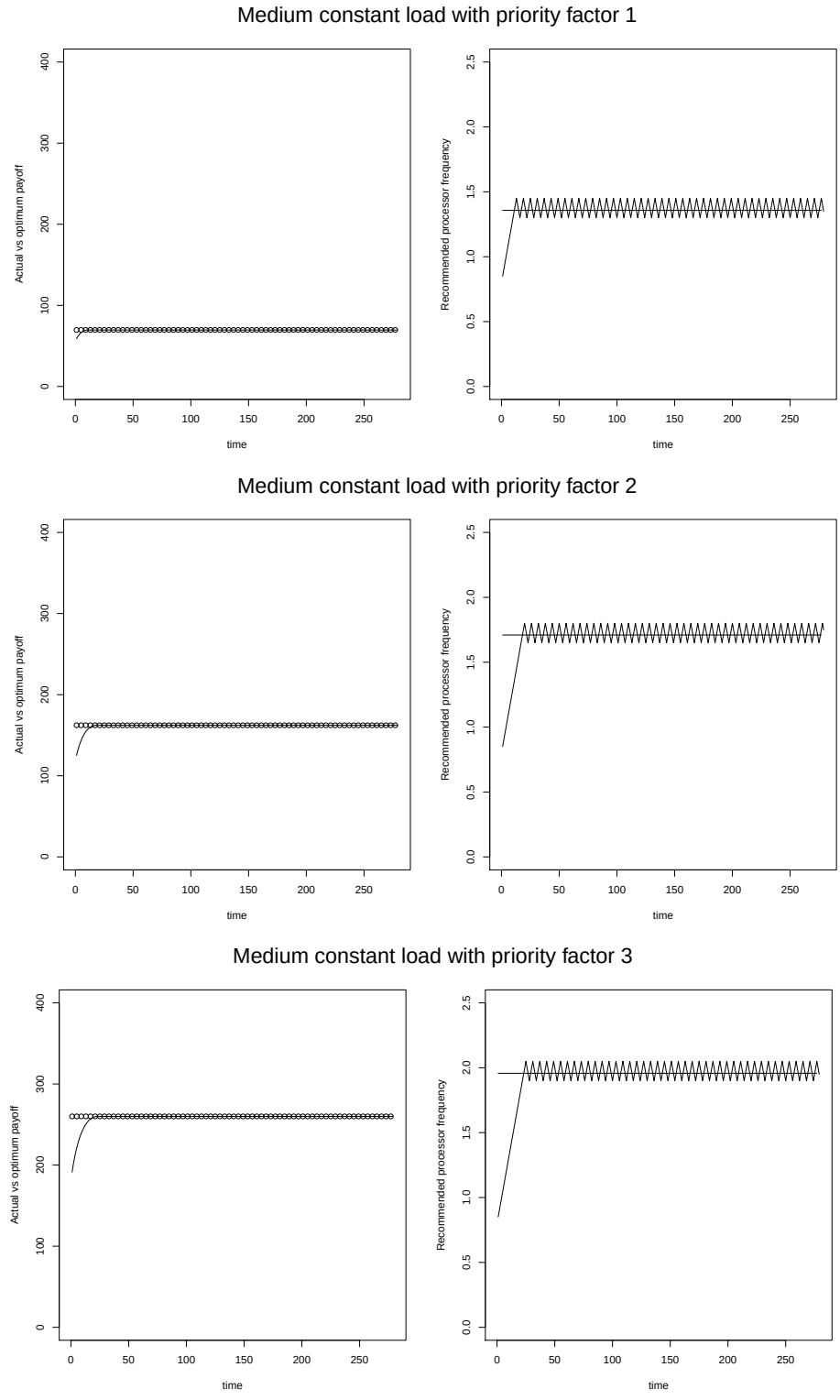

Figure 4.2: Actual vs. optimal payoff and optimal vs. recommended resource values with medium constant load (50%) and priority factors 1, 2 and 3. Optimum is shown with circles in the left graphs and with a straight line in the right graphs.

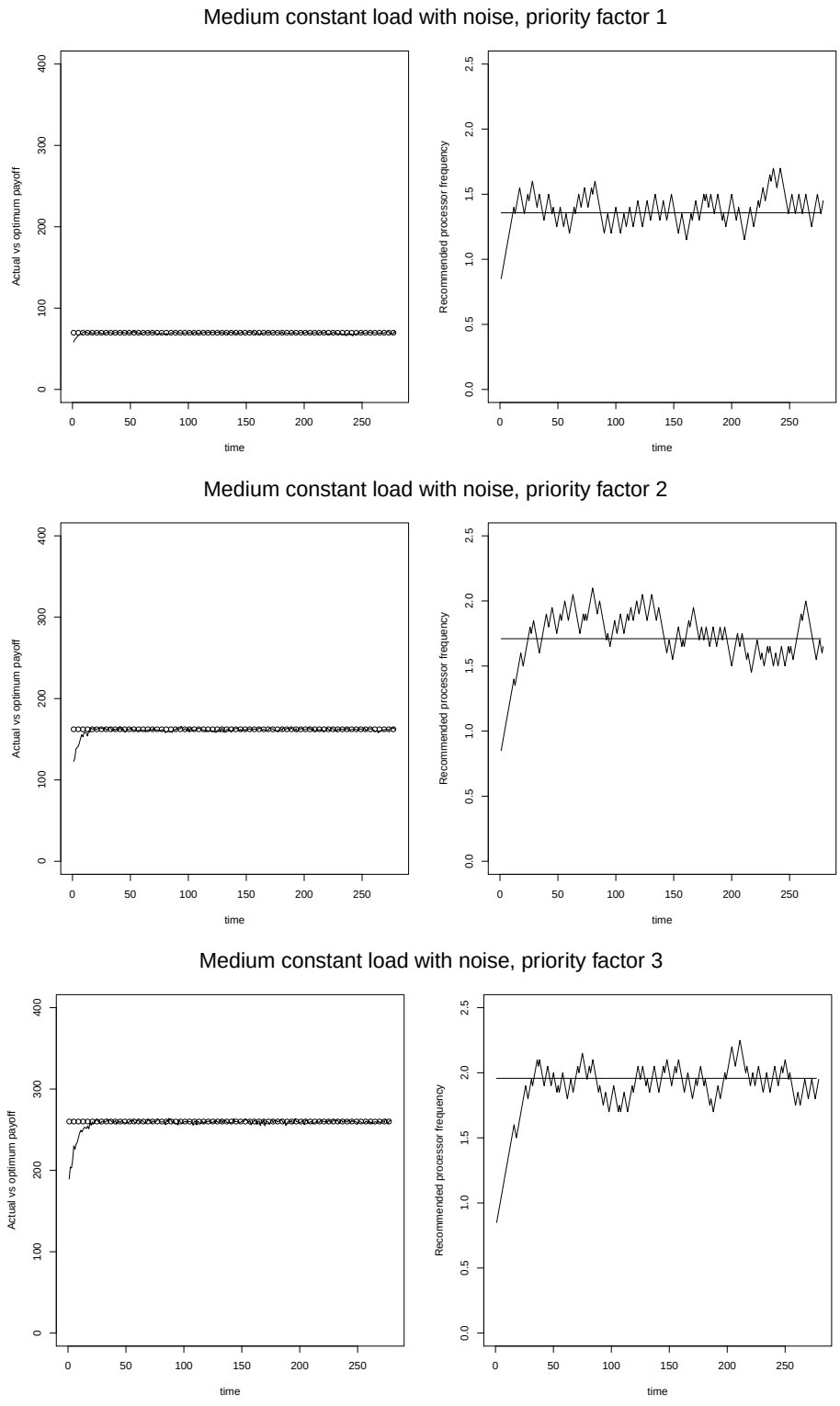

Figure 4.3: Actual vs. optimal payoff and optimal vs. recommended resource values with medium constant load (50%), added noise and priority factors 1, 2 and 3. Optimum is shown with circles in the left graphs and with a straight line in the right graphs.

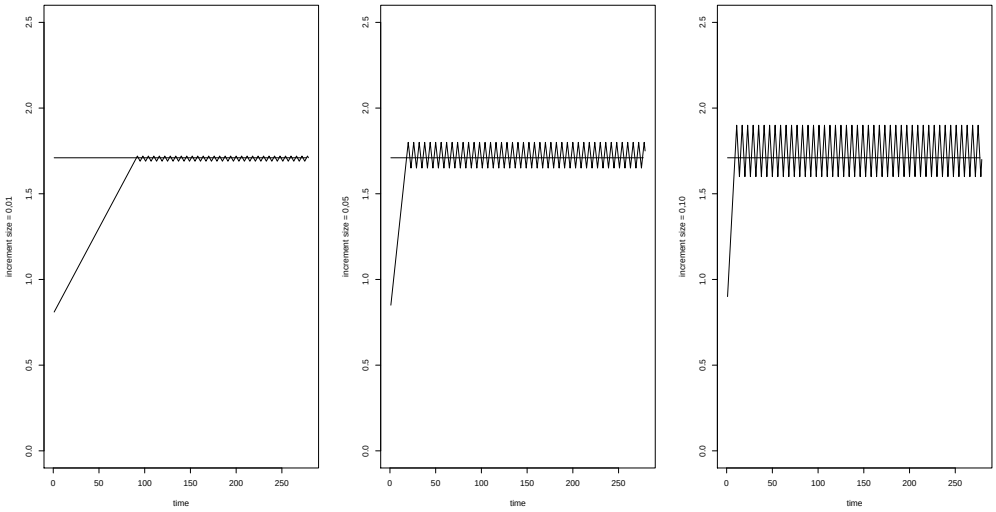

Figure 4.4: A smaller increment size on a constant load will give smaller oscillations around the theoretical optimum (the straight line), but also a longer settling time. Increment sizes from left to right: 0.01, 0.05 and 0.10.

4.1.4 Simulations with sinusoidal load

The graphs in Figure 4.5 show the results of the simulations with sinusoidal load. The first observation is that the graph of the recommended processor frequency seems to follow the theoretical optimum rather poorly. From the payoff graphs, we see that the actual payoff is slightly lower than the theoretical optimum. This is a phenomenon that Couch et al refers to as "undershoot," which means that convergence is a bit too slow due to an increment size that is not sufficiently large [11].

The graphs in Figure 4.6 show the effect of different increment sizes. The middle graph is the same graph as the one for priority factor 2 in Figure 4.5, while the left and right graphs show the recommended resource value with a smaller and a larger increment size, respectively. A smaller increment size gives even less accuracy (left graph), while a larger increment size gives less overshoot and a seemingly better behaviour, although it also give more oscillations.

The graph showing priority factor 2 looks worse than those with priority factors 1 and 3, due to its two equally high spikes. Looking a little bit closer, there is a second spike on all the graphs, placed where the theoretical optimum descends. Moreover, the descending optimum seems to hit the second spike just on the middle on all the graphs. This is because the recommended resource oscillates around the theoretical optimum, just like it does on the constant load graphs.

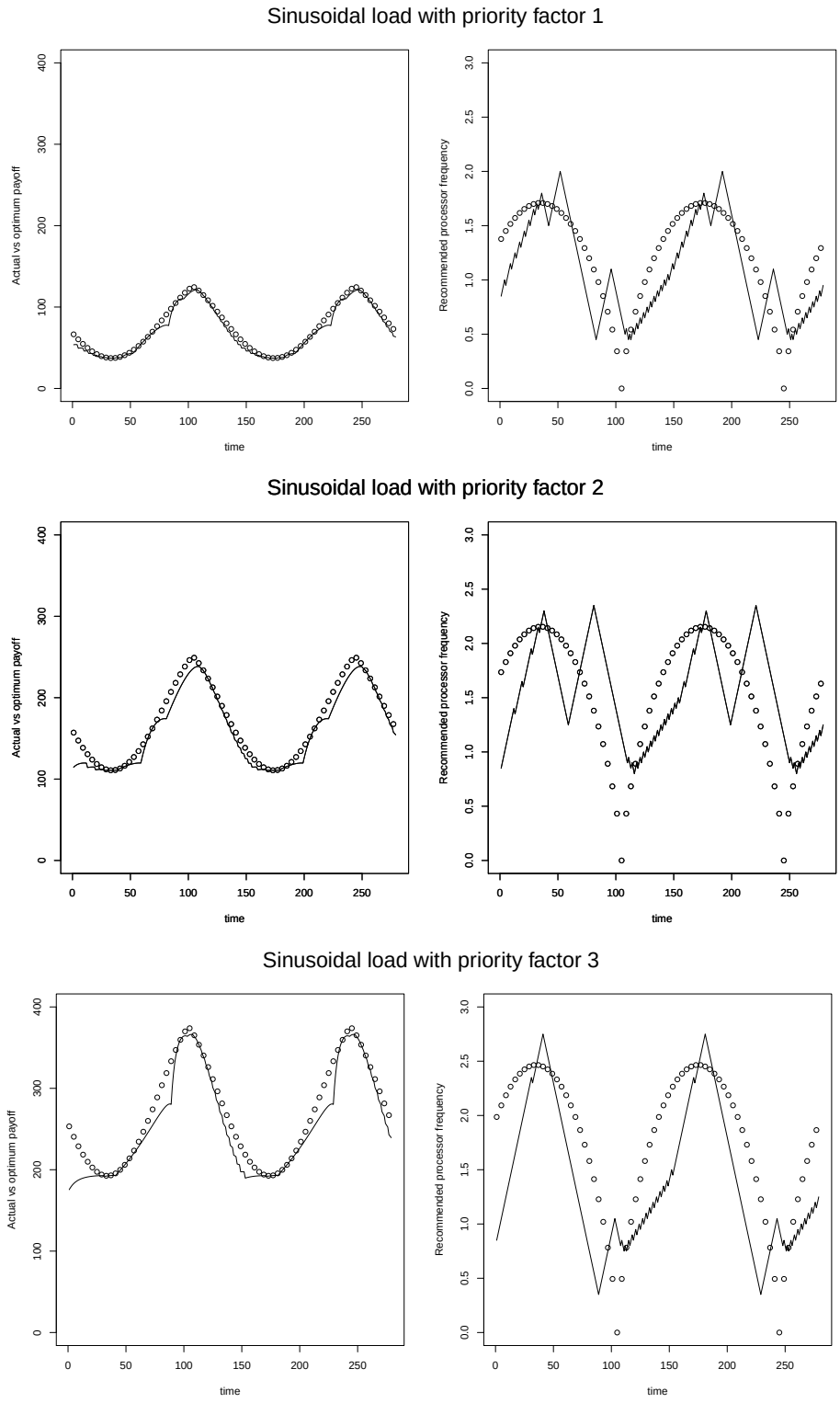

Figure 4.5: Actual vs. optimal payoff and optimal vs. recommended resource values with sinusoidal load and priority factors 1, 2 and 3. Optimum is shown with circles.

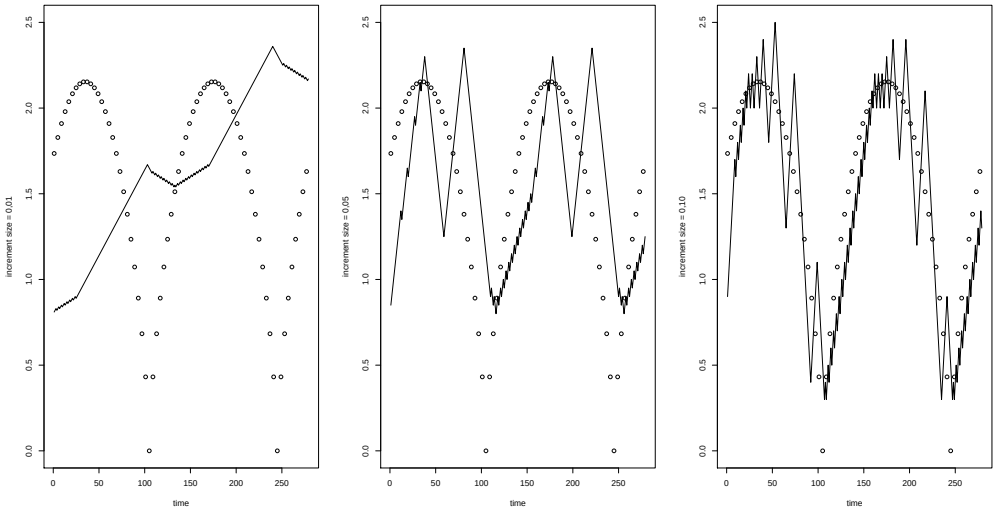

Figure 4.6: Effects of increment size on sinusoidal load. Higher increment gives shorter convergence time, but also more oscillations due to overshoot. Increment sizes from left to right: 0.01, 0.05 and 0.10.

However, the round sinusoidal shape of these graphs are harder to follow for a function with an oscillating and "spiky" behaviour than it is following a flat line. This makes the function look less accurate, while in fact it may not be with so very much. It therefore seems to be a coincidence that the priority 2 graph looks worse than the others, and that it has nothing to do with the priority factor as such.

4.1.5 Simulations with realistic load

The realistic load simulates an actual system load, with several characteristics of real load. It includes sections of both high and low load, steep and slow ascents and descents, and a short gigantic spike. The load is profiled in Figure 3.4 on page 30. The results of the simulations are presented in Figure 4.7. These are the frequency recommendations that will be used as input to the proof of concept implementation.

The graphs on the left side show that the payoff function is following the theoretical optimum quite well. There are no big differences between the different priority factors, except for a small magnification of errors due to the priority factor multiplication. The effect of the priority factors is better illustrated by the graph in figure 4.8. The horizontal line shows the mean of the resource recommendation. The mean increases with higher priority factors, and this behaviour confirms that the priority factor has the intended effect.

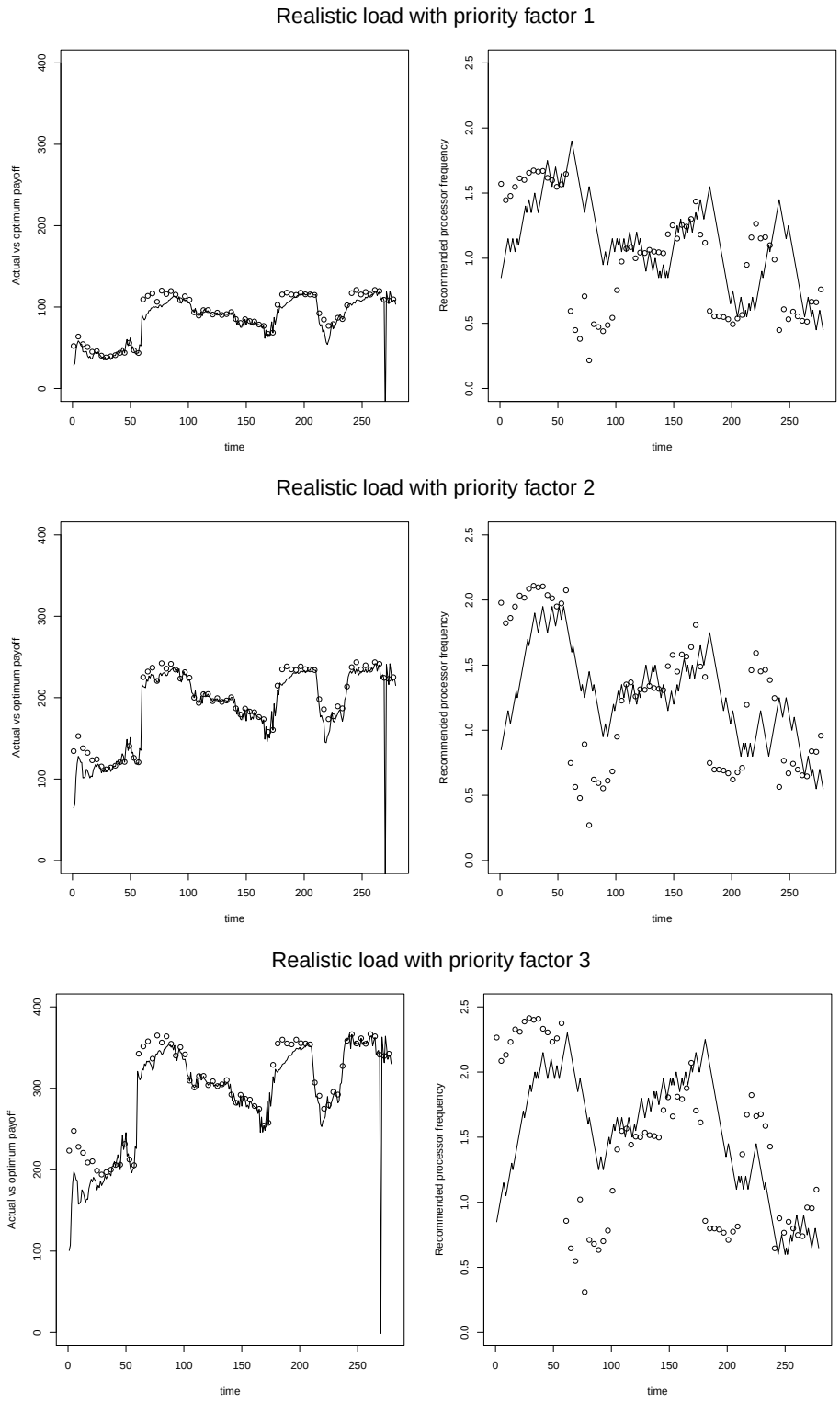

Figure 4.7: Actual vs. optimal payoff and optimal vs. recommended resource values with the realistic load and priority factors 1, 2 and 3. Optimum is shown with circles.

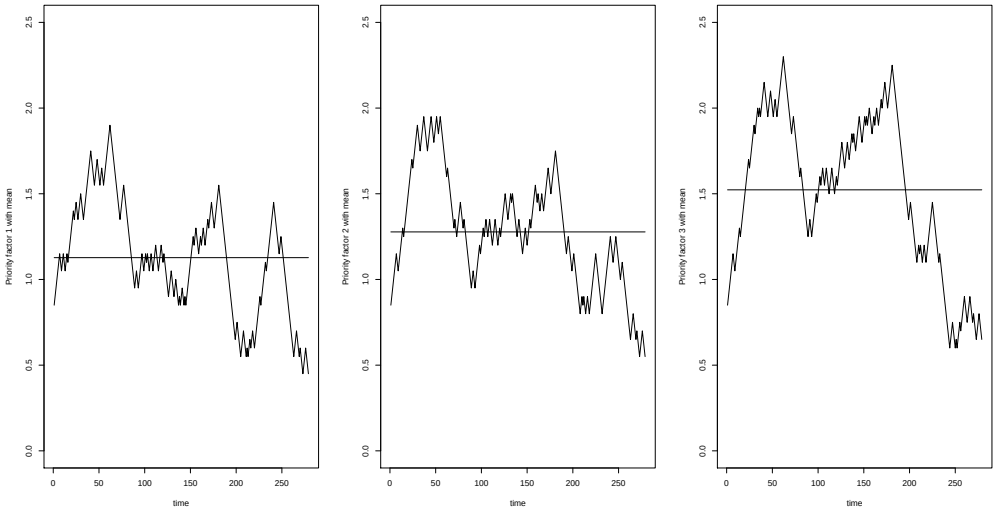

Figure 4.8: Resource recommendation for the realistic load, and priority factors from left to right are 1, 2 and 3. The horizontal line shows the mean, illustrating the effect of priority factors on the realistic load.

Several observations are made regarding the behaviour of the resource closure. Some settling time is needed for the recommended resource value to reach its peak value in the high-load area in the beginning. It can also be observed that the recommended resorurce value is not following the theoretical optimum quickly enough. This indicates an increment size that is less than optimal, causing slow convergence. As mentioned earlier, the chosen increment size is a compromise between sufficient reactivity and being small enough for the hysteresis function to avoid oscillations between discrete resource levels.

The graphs in figure 4.9 show the effect of increment size on the realistic load. A small increment size of 0.01, as is shown in the graph to the left, makes the recommended resource value worthless. A larger increment size of 0.10 will on the other hand improve convergence time, although it will also create more overshoot as we have discussed earlier.

Still, it is clearly performing better than the chosen increment size of 0.05. It seems that a larger increment size and high convergence tend to be preferred for input loads that have much variation, such as it was also observed with the sinusoidal load, while on more stable loads a smaller increment size will perform better, as observed with the constant load.

In the case of processor frequency scaling, a moderate increment size will not only be important for the hysteresis function to be effective, it will also make smoother changes. When the input load plummets, the graph shows us that the resource value follows more slowly. This smoothing behaviour is a desired property of the model, as long as it does not get too slow. A good illustration

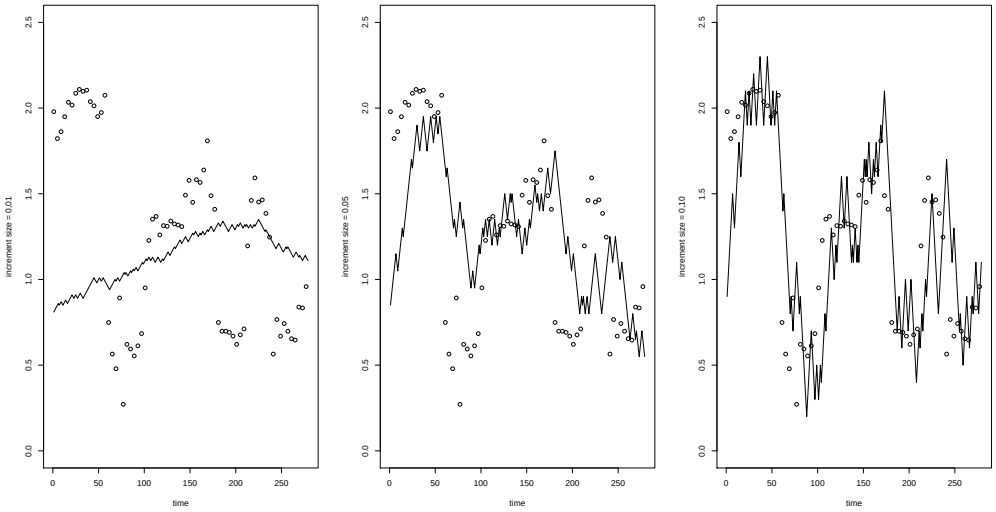

Figure 4.9: Resource recommendation for the realistic load with various increment sizes. Higher increment size gives shorter convergence time. Increment sizes from left to right: 0.01, 0.05 and 0.10.

of this is the gigantic spike at the end of the load period, which is not affecting the model any more than a lot smaller change in the same direction would have done.

4.2 Proof of concept

The proof of concept was run while manually observing the output from the script on the screen, as well as observing the changes made on the computer in real time using the "CPU frequency scaling monitor" applet for the Gnome desktop environment [8]. The observations confirmed that the script was working as intended, and that the processor frequency was set correctly. The output was also written to a log file.

The results of the proof of concept are presented in the graphs in Figure 4.10. The graphs confirm that the processor frequencies were set successfully, and as intended. However, one unwanted behaviour was observed. At all three priority levels, there were a few short spikes, indicating that the thresholds of the hysteresis function are not set sufficiently far apart. To confirm and to rectify this, a new run of the proof of concept was performed, this time with a downward threshold below the former threshold, as shown in table 4.1. The result in figure 4.11 shows that an increased span between the thresholds removed the remaining spikes. The new threshold span has also shifted some of the level changes.

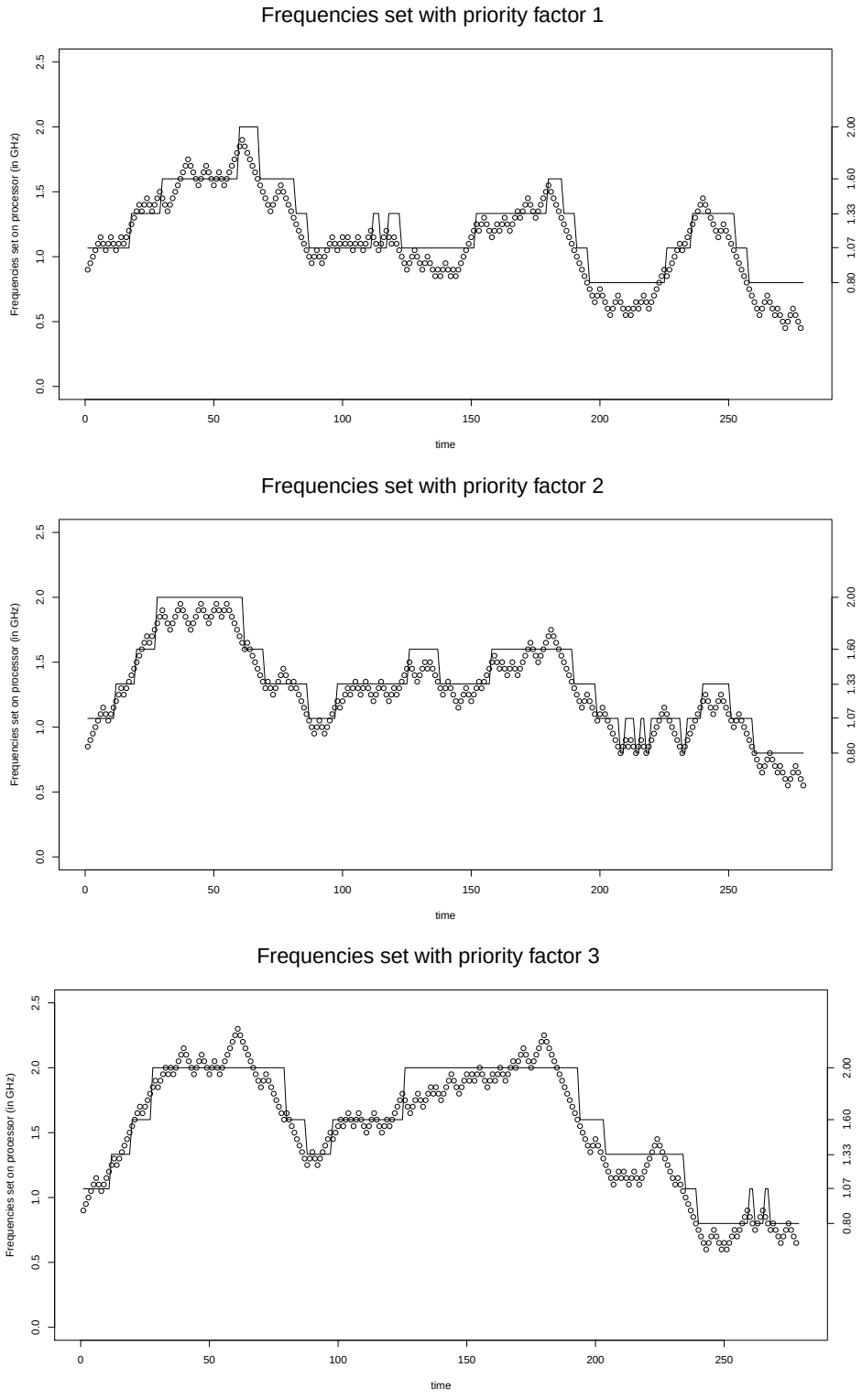

Figure 4.10: The frequency levels that were set on the processor during the proof of concept experiment, according to the recommendations made by the resource closure model. The lines show the frequencies that were set, and the recommendations are shown as a series of circles.

Table 4.1: Modified thresholds in the second run of the proof of concept.

Current freq. level	**Upward threshold**[1]	**Downward threshold**[2]
2.00 GHz	— [3]	1.44 GHz
1.60 GHz	1.76 GHz	1.20 GHz
1.33 GHz	1.46 GHz	0.96 GHz
1.07 GHz	1.18 GHz	0.72 GHz
0.80 GHz	0.88 GHz	— [4]

[1] Upward threshold is defined as 10% above current frequency level. This is as before.
[2] The modified downward threshold is now defined as 90% of the frequency level below the current level.
[3,4] The frequency is already at maximum/minimum level, and can not ascend/descend any further.

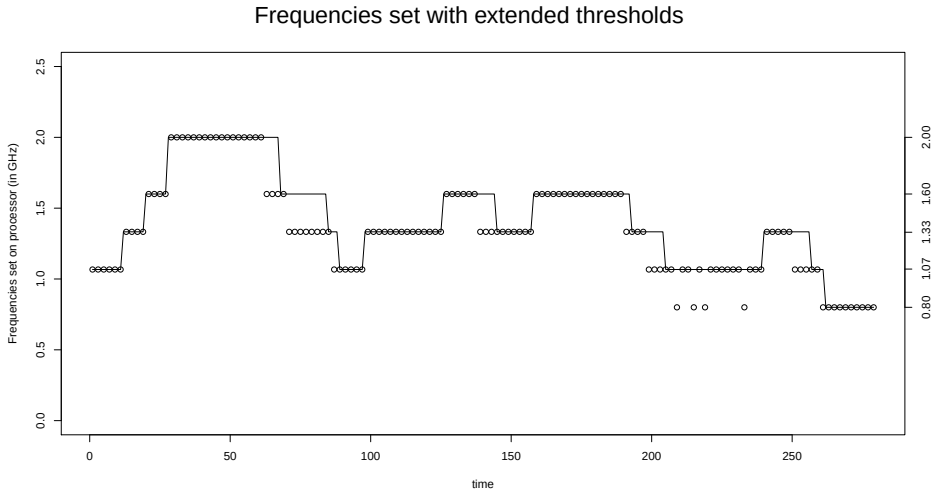

Figure 4.11: Frequencies set when extending the span between thresholds. The priority factor of this graph is 2, and the circles show the middle graph in Figure 4.10 for comparison. The spikes between 200 and 250 are now gone.

Table 4.2: Showing how close in percentage the different frequencies are set to the discrete theoretical optimal value. The percentages have been calculated after omitting initial settling time.

Inc. size / Threshold[1]	**Same level**[2]	**1 or less**[2]	**2 or less**[2]	**3 or less**[2]	**4 or less**[2]
0.05 / 0%	37,0%	71,5%	93,4%	99,5%	100,00%
0.10 / 0%	53,5%	86,0%	97,5%	99,5%	100,0%
0.10 / 20%	54,5%	86,5%	97,5%	100,0%	100,0%

[1] The rows are corresponding with the graphs in Figure 4.12.
[2] The columns show how much of the time the frequency set corresponds with the discrete optimal frequency, and when it is 1 level or less away, 2 levels or less away, and so on.

4.2.1 Optimal frequency level

To get a better picture of how efficiently the proof of concept performs, it is interesting to compare how it is controlling the processor with the current model, to how it would control the processor if using the theoretical optimum as its input data. This comparison is shown in Figure 4.12 and Table 4.2. These tests are all made with priority level 2.

The first graph shows the frequencies set when increment size is 0.05 and the span between the threshold levels are 0%, in other words, when the hysteresis function has been disabled. From before we know that the increment size is a bit too low to be sufficiently reactive, and without the hysteresis function, we get unwanted spikes with rapid frequency level change. Although the frequencies are following the general trend of the optimal frequency levels, it is clearly converging too slowly. This is easily observed in Table 4.2, where the frequency only follows the optimum 37% of the time.

When increasing the increment size, convergence time is improved, and we get recommendations that lie more closely to the optimum, being 2 levels or more away from the optimum less than 15% of the time. However, since we have still not enabled the hysteresis function, there are still a lot of unwanted spikes. In the last graph, the hysteresis function is activated with a span between thresholds of 20%. As expected, this eliminates some of the spikes, but not all, indicating that it either needs an even broader span between thresholds or a smaller increment size. This illustrates the conflict between these two parameters.

More interestingly, however, from the table it can be read that the hysteresis function also slightly improves how closely the recommendations from the closure follow the theoretical optimum. This indicates that the hysteresis function is not degrading the exactness of the closure, which is counterintuitive considering that the hysteresis function prevents the resource from immediately reaching its desired state.

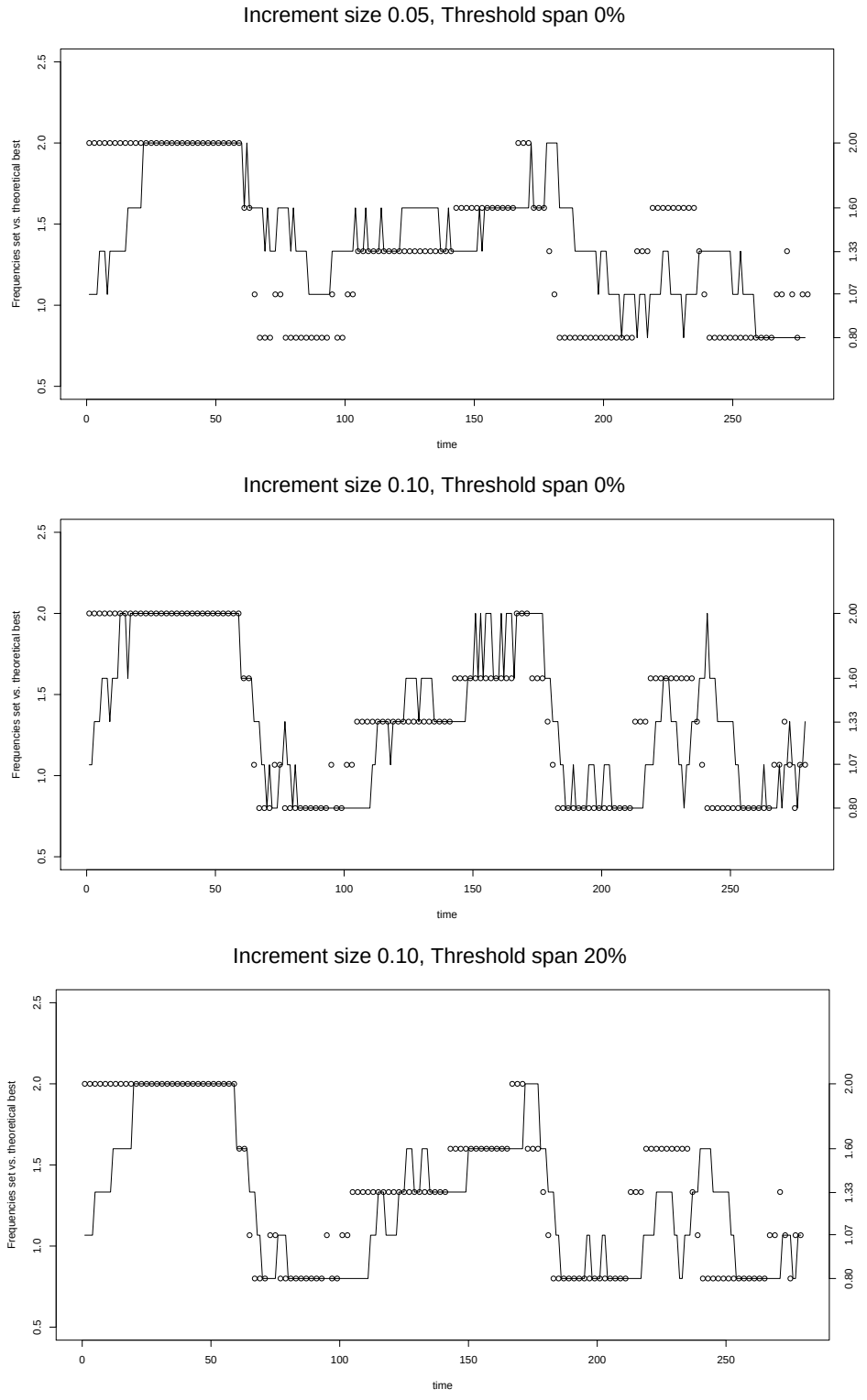

Figure 4.12: Showing actual processor frequencies set during the proof of concept experiment vs. frequency levels set when using the the theoretical optimum as input. The discrete optimum values are shown as circles in the graphs.

Table 4.3: Mean power consumption of proof of concept implementation with the realistic load.

Priority factor	**Mean power cons.**	**% of full capacity**
Priority 1	15.2 W	56%
Priority 2	17.1 W	63%
Priority 3	20.2 W	75%
Full capacity	27 W	100%

4.2.2 Power reduction

The purpose of the proof of concept is to enable a reduction in power consumption. It is not at this stage possible to make any exact calculations on the amount of power saved, nor to make any evaluation of the effectiveness of this model versus existing frequency scaling methods. However, some estimations can be made, based on the power specifications of the processor used in the experiment and the estimations of voltage and power dissipation values (noted in table 3.1 on page 31). Using these numbers, an estimation can be made about the mean power consumption of the processor during the proof of concept experiment. The results of these calculations are shown in table 4.3.

Chapter 5

Discussion and conclusion

This chapter will discuss the results of this masters thesis, as well as implications of the approach and design choices made. The discussion will lead up to the conclusion, which will give a final evaluation of the work performed during the course of this thesis, and suggest future research that can spring from the work herein.

5.1 Review of approach and design

The work with this thesis has primarily been based on theoretical development and simulations. Although recent work on resource closure operators has been very promising, the concept is still at a very theoretical level. Many remaining issues has to be solved before the concept is mature enough and ready to be implemented in a production environment. A theoretical approach has therefore been necessary also for this thesis.

Although a few proof of concept implementations of closures have been performed in the field of configuration management, none has earlier been attempted specifically for the resource closure model. The proof of concept implementation of processor frequency scaling in this thesis is the first of its kind in this regard.

There have been many design choices that had to be made along the way. Even when topic was chosen, it was not obvious how to narrow it down and what the outcome would be. This thesis is a good example of how the road is being made as you go along. The original idea was to investigate how the resource closure model could be used for resource management in a cloud computing data center. While researching challenges in cloud computing resource management, the issue of power consumption and power management was encountered repeatedly. The impact it has on the environment also made its importance much more evident, and so the thesis soon turned its focus entirely in this direction.

The next big choice was which field of research within power management to concentrate on. Two areas immedieately seemed more interesting than others, and that was turning off unutilised equipment and downscaling of processor frequency levels. Both areas were considered, since these methods also complement each other very well. It was quickly apparent however, that the time constraint would not allow for both. Frequency scaling was chosen, not only because it seemed like the most doable approach in terms of making a proof of concept implementation, but also because it had interesting research problems that challenged the current resource closure model. The issues of a non-linear cost function, priority factors in the value function or discrete resource level steps has never been fully explored in earlier research.

The desire for designing a proof of concept was present long before the case was chosen, and this wish was in many ways the driving force behind many of the design choices made. The proof of concept was successfully conducted, although if time had allowed, it would have been desirable to be able to develop an even more realistic implementation. Time is always a restricting factor in a masters thesis, and especially for a one semester thesis like this one. However, the time has proven to be sufficient to do thorough research of the chosen topic, to develop new additions to the resource closure model, to perform the necessary simulations and to make a proof of concept implementation that successfully controls processor frequency levels on a computer using input from the resource closure.

5.2 Review of results

The results from this thesis includes both the additions made to the original resource closure model, and the results of the simulations of the new model, as presented in Chapter 4. It also includes the proof of concept implementation and the results showing the feasibility of using such a model in a case of dynamic frequency scaling.

5.2.1 Validity, reliability and reproducibility

Theories and concepts developed during this thesis have been implemented in a simulation environment in *R*. It is the same environment as used by Couch et al in their research on resource closure operators, and the validity and reliability of the environment should therefore be considered very high.

The results of the simulations are based on various input data. Constant and sinusoidal load are easily generated by *R*. Constant load gives an interesting insight in how the model performs without changes. According to Couch et al, the model does not perform optimally when the value function is not changing [11]. Sinusoidal load is what is used by Couch et al when testing their model,

and profiles of real load-data sometimes resemble sinusoidal load rather closely [13].

The realistic load input has been partially profiled on a real case of server load, harvested from computer usage during a first year graduate course on system administration. The load is static, and is not affected by the resource closure as it would have been in a functioning feedback loop. However, the load, and any other factor that has an influence on the load, is in any case unknown to the closure, so for simulation purposes and for the purpose of a simple proof of concept implementation, this is not of vital importance.

The simulations made in this thesis are easily reproduced, provided that one has access to the simulation environment in *R*. Since the original source code is not a product of this master thesis, it can not be included. However, for those interested in making further research on the resource closure model, the *R*-code could probably be acquired by contacting Professor Couch. The Perl script implementing the proof of concept is provided as an appendix.

5.2.2 Additions to the model

The resource closure model design including the new additions is shown in figure 3.1 on page 27.

Non-linear cost function

The non-linear cost function was necessary to apply to the model in order to implement it in the case of frequency scaling, because the relation between frequency and power consumption is quadratic. Simulations of the model with sinusoidal load and a quadratic cost function showed, when compared with the re-calculated theoretical optimum, that it is equally efficient as a linear cost function. This was an important finding and vital for further research into the chosen case.

Value function priority factor

The value function priority factor is an attempt to address the notion that not all workloads are equally important.[1] One of the strengths of this method is its simplicity, both programmatically and cognitively. Simulations also showed that the priority factor had more impact on the total value when the original value was high than when it was low. This behaviour is desired, because it makes higher resource levels more attainable when dealing with higher priority workloads, without affecting the resource recommendation as much on lower

[1] Or as George Orwell would have put it, "All workloads are equal, but some workloads are more equal than others."

loads. When dealing with small loads, low resources will be sufficient, even if the priority of the workload is high.

The shortcomings of the priority factor is that it does not take into consideration absolute SLA requirements, it only gives a possibility to say that "this workload is more important, please take it into consideration when choosing the resource level." Moreover, it can not differentiate between different priority workloads on the same server. One could, however, imagine a front-end switching of the different workloads, sending them to the right priority servers, combined with a dynamic control of how many high-priority servers are needed.

Resource level steps

The resource level hysteresis function presented in algorithm 1 on page 26 is presented as a way of dealing with limited and discrete resource levels, to avoid oscillation between two resource levels when the resource values stay close to the limit value between two such levels. As we know from the simulations, the recommendations from the closure has an oscillating behaviour when load is constant, and makes this unwanted behaviour very likely to happen.

When running the proof of concept, we could see that the size of the area between the thresholds is an important factor to make the hysteresis function work properly. Having a threshold width of 10% was not sufficient to remove all the spikes in that particular workload, but with a threshold width of approximately 20%, all spikes were gone (see figure 4.11).

5.2.3 Optimisation

The resource closure model has several parameters, the original model has increment size and window size, and the new model also has priority level and the threshold span of the hysteresis function. Tuning these parameters is very important to make the model perform as desired. However, tuning the model for optimal behaviour has not been a prioritised task in this master thesis, because such a tuning is in any case very dependent on the environment of its application and on the situation, which may also change over time. It is therefore not of any vital interest for a proof of concept, beyond that it should perform good enough to be functioning adequately.

The topic has, however, been shown some consideration, and a comparison between the performance of the proof of concept experiment and a theoretical optimal frequency value has been performed. The results are presented in Table 4.2 and Figure 4.12, and described in the results chapter.

An interesting finding was that the hysteresis function is not degrading the performance of the resource recommendations, and the percentages even show a slight improvement. This is interesting because the purpose of the hysteresis

function is to prevent the resource recommendations from oscillating between discrete resource levels, and it does this by to a certain degree preventing the resource level from reaching its desired state.

It has to be noted that the theoretical optimum is not necessarily fully optimal in reality, and that a 100% correspondence is probably not possible, nor is it necessarily desired. One might say that the theoretical optimum is how the resource closure model would perform with no convergence time, where the resource level is optimal at all times. For one thing, this is not achievable without full knowledge of the load, which is one of the presumptions that we do not have. Moreover, a little convergence time is desirable for smoothening the graph to avoid spikes, which has been shown to work.

The choice of increment size has also shown to be dependent on the typical load profile of the system. A system with a load with much variation over short time periods tend to need a larger increment size, while stable loads will perform better with smaller increment size. The hysteresis function is also dependent on a small increment size, and this might lead to a conflict in a system with high variation in its load, and a non-optimal compromise will have to be to be made.

5.2.4 Power reduction

One of the goals of this master thesis was to develop a model that could enable a reduction of power consumption in a computer system, and the method chosen to achieve this was dynamic processor frequency scaling. The model was developed and implemented in a proof of concept perl script that successfully was able to adjust the processor frequency based on input from the resource closure.

Although there has not been performed any measurements to verify any reduction in power consumption, estimations (presented in table 4.3) indicates a near halving of the power consumption on the lowest priority level, and a reduction by a quarter on the highest priority level. This is a strong indication that a reduction in power consumption is achievable using a resource closure model.

It has to be noted, though, that the numbers presented in this thesis are only indications, as they are calculated from estimates; and also that they are only valid for the load used in the experiment. In addition, they do not take all factors into consideration, such as continually adjusting for the power consumption with the exact load.

5.3 Conclusion

The goal of this thesis has been to investigate and evaluate the possibility of using the resource closure model for power management in a computer sys-

tem, and to make further developments to the model in order to be able to make a proof of concept implementation in a specific case of autonomic power management.

During the course of this thesis, the resource closure model has been further developed, and three new additions to the model have been presented. These additions have been implemented in a successful proof of concept experiment that has been able to show that the resource closure model is able to make recommendations that can be used to control processor frequency in accordance with incoming load. The simulations and proof of concept conducted have produced interesting and encouraging results, that are presented in this document.

Estimations has been made to evaluate the expected effect of implementing the resource closure model in a production environment. There are uncertainties in the presented estimates, but after taking this into consideration there is still a strong indication that the use of a resource closure model is a viable approach for autonomic power management. Accordingly, power management seems to be a good field of study for further development of the resource closure model.

5.3.1 Contributions

The main achievements of this masters thesis is its contributions to the study of resource closures. Most significantly is the development of three new additions to the resource closure model, namely, the non-linear cost function, value function priority factor and hysteresis function for discrete and limited resource levels. Although they have been developed to solve issues related to the chosen implementation, the additions are generic in nature and conceivably should be usable in several other cases where a resource closure model could be implemented.

5.3.2 Future work

The results from this masters thesis opens up several prospects for future work. First and foremost, there are still several unsolved general issues concerning the resource closure model. Couch et al mention several issues that has not been addressed in this thesis [11, 10]. In addition, from this thesis, the suggested concept of priority factors is not sufficient in a case of absolute SLA requirements and needs to be either improved or replaced. Also, how to differentiate between different workloads is an unsolved challenge. In this thesis, it has been proved that a non-linear cost function is possible. However, other cost functions, for instance an exponential function, have not been attempted.

In the proof of concept, the processor frequency levels were successfully controlled using recommendations from the resource closure. However, this was achieved by using a simulated load read in from a file. The next natural step would be to test the proof of concept further, this time by reading in real time

processor load directly. The biggest advantage to this would be that it enables the recommendations to influence the load directly through the feedback loop.

After completing a proof of concept with a real time load, the next step could be to test the resource closure model on several servers simultaneously in conjunction with a load balancer. Also, measuring actual energy savings would be natural to do in such an experiment. Eventually, a working proof of concept needs to be tuned and optimised. At this stage it would also be natural to compare the resource closure model with other methods of frequency scaling, such as the ones described by Sharma et al [42], Lu et al [34], and Kusic et al [32].

Bibliography

[1] Advanced Micro Devices Inc. *AMD® Athlon™ 64 Processor Power and Thermal Data Sheet*, 2006, Rev. 3.51.

[2] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. *Power aware computing*, chapter 14. The case for power management in web servers. Series in computer systems. Springer, 2002.

[3] Megan Bray. Review of computer energy consumption and potential savings. White paper, December 2006. Available at `http://www.dssw.co.uk/research/`.

[4] Dominik Brodowski. *Linux CPUFreq*. Available at `http://www.kernel.org/doc/Documentation/cpu-freq/index.txt`.

[5] Dominik Brodowski. *Linux CPUFreq-utilities (cpufrequtils)*. Available at `http://www.kernel.org/pub/linux/utils/kernel/cpufreq/cpufrequtils.html`.

[6] Mark Burgess. *Analytical Network and System Administration*. John Wiley & Sons, Ltd, 2004. ISBN 0-470-86100-2.

[7] Mark Burgess. Configurable immunity for evolving human-computer systems. *Science of Computer Programming*, 51(3):197–213, 2004.

[8] Carlos Garcia Campos and Davyd Madeley. *CPU Frequency Scaling Monitor Manual*. GNOME Foundation, August 2005.

[9] Alva L. Couch. Summary of the third workshop on hot topics in autonomic computing. *;login: Magazine*, 33(5):112–113, October 2008. Available at `http://www.usenix.org/publications/login/2008-10/index.html`.

[10] Alva L. Couch, Mark Burgess, and Marc Chiarini. Management without (detailed) models. In *Proceedings of ATC '09*, pages 75–89. International Conference on Autonomous Infrastructure, Management and Security, 2009.

[11] Alva L. Couch and Marc Chiarini. Dynamics of resource closure operators. In *Proceedings of AIMS '09*, pages 28–41. International Conference on Autonomous Infrastructure, Management and Security, 2009.

[12] Alva L. Couch, John Hart, Elizabeth G. Idhaw, and Dominic Kallas. Seeking closure in an open world: A behavioral agent approach to configuration management. In *Proceedings of LISA '03*, pages 125–148. Large Installation Systems Administration Conference, 2003.

[13] Stephen G. Eick, William A. Massey, and Ward Whitt. $M_t/G/\infty$ queues with sinusoidal arrival rates. *Management Science*, 39(2):241–252, February 1993.

[14] Mark E. Femal and Vincent W. Freh. Boosting data center performance through non-uniform power allocation. In *Proceedings of ICAC '05*, pages 250–261. IEEE International Conference on Autonomic Computing, 2005.

[15] Wu-Chun Feng. Making a case for efficient supercomputing. *ACM Queue*, October 2003.

[16] Robert R. Harmon and Nora Auseklis. Sustainable it services: Assessing the impact of green computing practices. In *Proceedings of PICMET '09*, pages 1707–1717. Portland International Center for Management of Engineering and Technology, 2009.

[17] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[18] Paul Horn. Autonomic computing: Ibm's perspective on the state of information technology. Online, October 2001. Available at `http://researchweb.watson.ibm.com/autonomic/manifesto/autonomic_computing.pdf`.

[19] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing - degrees, models, and applications. *ACM Computing Surveys*, 40(3), August 2008.

[20] Intel Corp. *Enhanced Intel® SpeedStep® Technology for the Intel® Pentium® M Processor*, March 2004. Available at `http://www.intel.com/design/intarch/pentiumm/docs_pentiumm_90nm.htm`.

[21] Intel Corp. *Intel® Pentium® M Processor with 2-MB L2 Cache and 533-MHz Front Side Bus - Datasheet*, July 2005. Available at `http://www.intel.com/design/intarch/pentiumm/docs_pentiumm_90nm.htm`.

[22] Internet World Stats. Internet usage statistics. Online. Available at `http://www.internetworldstats.com/stats.htm`. Accessed May 19, 2010.

[23] Nagarajan Kandasamy, Sherif Abdelwahed, and John P. Hayes. Self-optimization in computer systems via online control: Application to power management. In *Proceedings of ICAC '05*, pages 54–61. IEEE International Conference on Autonomic Computing, 2004.

[24] Jeffrey O. Kephart. Research challenges of autonomic computing. In *Proceedings of ICSE '05*, pages 15–22. International Conference on Software Engineering, 2005.

[25] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer Magazine*, 36(1):41–50, January 2003.

[26] Bithika Khargharia, Salim Hariri, Wael Kdouh, Manal Houri, Hesham El-Rewini, and Mazin Yousif. Autonomic power and performance management of high-performance servers. In *Proceedings of IPDPS '08*, pages 1–5. IEEE International Symposium on Parallel and Distributed Processing, 2008.

[27] Bithika Khargharia, Salim Hariri, Ferenc Szidarovszky, Manal Houri, Hesham El-Rewini, Samee Ullah Khan, Ishfaq Ahmad, and Mazin S. Yousif. Autonomic power & performance management for large-scale data centers. In *Proceedings of IPDPS '07*, pages 1–8. IEEE International Symposium on Parallel and Distributed Processing, 2007.

[28] Bithika Khargharia, Salim Hariri, and Mazin S. Yousif. Autonomic power and performance management for computing systems. *Cluster Computing*, 11(2):167–181, June 2008.

[29] Bithika Khargharia, Salim Hariri, and Mazin S. Yousif. An adaptive interleaving technique for memory performance-per-watt management. *IEEE Transactions on Parallel and Distributed Systems*, 20(7):1011–1022, July 2009.

[30] Jonathan G. Koomey. Estimating total power consumption by servers in the u.s. and the world. Technical report, Lawrence Berkeley National Laboratory and Stanford University, 2007.

[31] Patrick Kurp. Green computing. *Communications of the ACM*, pages 11–13, October 2008.

[32] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. In *Proceedings of ICAC '08*, pages 3–12. IEEE International Conference on Autonomic Computing, 2008.

[33] Peter J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, January 1964.

[34] Zhijian Lu, Jason Hein, Marty Humphrey, Mircea Stan, John Lach, and Kevin Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Proceedings of CASES '02*, pages 156–163. International Conference on Compilers, Architecture and Synthesis for Embedded Systems, 2002.

[35] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[36] Justin Moore, Jeffrey S. Chase, and Parthasarathy Ranganathan. Weatherman: Automated, online, and predictive thermal mapping and management for data centers. In *Proceedings of ICAC '05*, pages 155–164. IEEE International Conference on Autonomic Computing, 2006.

[37] Vittorio E. Pareto and Marcos P. Pareto. The urban component of the energy crisis. *Urbanistica*, August 2008. Available at `http://ssrn.com/abstract=1221622`.

[38] Trevor Pering, Tom Burd, and Robert Brodersen. Dynamic voltage scaling and the design of a low-power microprocessor system. In *Proceedings of Power-Driven Microarchitecture Workshop '98*. International Symposium on Computer Architecture, 1998.

[39] Johan Pouwelse, Koen Langendoen, and Henk Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of ACM SIGMOBILE '01*. ACM International Conference on Mobile Computing and Networking, 2001.

[40] Judy A. Roberson, Gregory K. Homan, Akshay Mahajan, Carrie A. Webber, Bruce Nordman, Richard E. Brown, Marla McWhinney, and Jonathan G. Koomey. Energy use and power management in new personal computers and monitors. Technical report, Lawrence Berkeley National Laboratory, 2002. Available at `http://www.osti.gov/energycitations/product.biblio.jsp?osti_id=799557`.

[41] Steven Schwartzberg and Alva L. Couch. Experience in implementing an http service closure. In *Proceedings of LISA '04*, pages 213–230. Large Installation Systems Administration Conference, 2004.

[42] Vivek Sharma, Arun Thomas, Tarek Abdelzaher, Kevin Skadron, and Zhijian Lu. Power-aware qos management in web servers. In *Proceedings of RTSS '03*. IEEE International Real-Time Systems Symposium, 2004.

[43] Mujtaba Talebi and Thomas Way. Methods, metrics and motivation for a green computer science program. In *Proceedings of SIGCSE '09*, pages 362–366. ACM Technical Symposium on Computer Science Education, 2009.

[44] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: Towards a cloud definition. *Computer Communication Review*, January 2009.

[45] W. N. Venables, D. M. Smith, and the R Development Core Team. *An Introduction to R, Version 2.10.1 (2009-12-14)*. Available at `http://www.r-project.org`.

[46] Linda Wilbanks. Green: My favorite color. *IT Professional*, pages 62–63, November/December 2008.

[47] Ning Wu and Alva Couch. Experience implementing an ip address closure. In *Proceedings of LISA '06*, pages 119–130. Large Installation Systems Administration Conference, 2006.

Appendix A

Proof of concept perl script

This perl script implements a proof of concept of the resource closure model. Some of the logging and print-statements have been removed to be more readable.

```
#!/usr/bin/perl

##########################################################################
# This perl script is written for the Master thesis
# "Autonomic Power Management using a Resource Closure Model"
# at University of Oslo and Oslo University College 2010.
# Author: Bengt Olav Olsen - s154215@stud.iu.hio.no
##########################################################################

# Additional modules

use strict;
use warnings;

# Variable declarations

# File containing available frequency levels on system
my $freqfile = "/sys/.../cpufreq/scaling_available_frequencies";
# File containing start-frequency of system
my $startfile = "/sys/.../cpufreq/scaling_cur_freq";
# Input-file for load
my $inputfile = "/.../resource.txt";
# Output-file with resource levels
my $outputfile = "/.../output.txt";

my $sleep = 1;      # Iteration sleep time in seconds
```

```
my @frequencies;    # Array for available frequency levels
my @resources;      # Array for resource-values from closure
my @setfreqs;       # Array for frequencies set by the closure
my $i = 1;          # Iteration counter
my $line;           # For storing each line from file
my @items;          # For storing items from line

my $resource = 0;   # Current resource parameter (R_rec)
my $deltaR = 0;     # Delta R sign 0 = negative, 1 = positive

my $frequency = 0; # Cur. freq. level as index of @frequencies
my $f_max = 0;      # Maximum available frequency level (R_max)
my $f_min = 0;      # Minimum available frequency level (R_min)

my $threshold = 0; # Threshold for changing frequency level
my $threshold_up = 1.1;   #110% # Upbound thrsh in %-decimal
my $threshold_down = 1.0; #100% # Downbound thrsh in %-decimal

# Fetch available frequencies

open(FREQ, "$freqfile") or die "Can't open $freqfile: $!";
while(<FREQ>) {
  @frequencies = split(/\s/);   # Read frequencies into array
}
close FREQ;

# Fetch resources

open(INPUT, "$inputfile") or die "Can't open $inputfile: $!";
foreach $line (<INPUT>) {
  chomp($line);

  # Put contents of the line in an array, because
  # format of output from R is 5 items per line
  @items = split(/\s/, $line);

  # Convert items to right format
  for (my $j = 0; $j < scalar(@items); $j++) {

    # If resources are in GHz format, convert to Hz
    if($items[$j] < 10) {
      $items[$j] = $items[$j]*1000000;
    }
    # If resources are in MHz format, convert to Hz
    if($items[$j] < 10000) {
      $items[$j] = $items[$j]*1000;
    }
```

```
  }

  push(@resources, @items);      # Read resources into array
}
close(INPUT);

# Set frequency, f_min and resource

$f_min = @frequencies-1;
$frequency = $f_min;
$resource = $frequencies[$frequency];

open(OUTPUT, ">$outputfile") or die "Can't open $outputfile: $!";

# Run loop for adjusting frequency levels

while(@resources) {

  # Read new resource level
  my $new_resource = shift(@resources);

  # At first iteration, set processor frequency directly
  # according to resource
  if($i == 1) {
    my $set_freq = $frequencies[$f_min];
    for(my $j = 0; $j < @frequencies; $j++) {
      if($frequencies[$j] >= $new_resource) {
        $set_freq = $frequencies[$j];
        $frequency = $j;
      }
    }
    system("cpufreq-set", "-f", "$set_freq");
    print(OUTPUT "Startup frequency: $set_freq\n");
  }

  print(OUTPUT "\nIteration $i\n");
  print(OUTPUT "New resource value: $new_resource\n");
  $i++; # Increment iteration count

  # Check if deltaR increment is positive, negative or unchanged
  if($new_resource < $resource) {$deltaR = -1;}      # negative
  elsif($new_resource > $resource) {$deltaR = 1;}    # positive
  elsif($new_resource == $resource) {$deltaR = 0;}   # unchanged

  # Set new resource
  $resource = $new_resource;
```

```
# Resource Level Hysteresis Function

if($deltaR > 0) {              # Delta R increment was positive
  # Calculate threshold
  $threshold = $frequencies[$frequency]*$threshold_up;
  print(OUTPUT "Increment threshold: $threshold\n");
  # Check if resource has reached threshold
  if($resource > $threshold) {
    # Check if frequency is already maximum
    if($frequency != $f_max) {
      print(OUTPUT "Frequency $frequencies[$frequency] changed to ");
      $frequency--;            # Adjust frequency level upwards
      print(OUTPUT "$frequencies[$frequency]\n");
      system("cpufreq-set", "-f", "$frequencies[$frequency]");
    } else {
      print(OUTPUT "Already at highest available frequency\n");
    }
  } else {
    print(OUTPUT "Frequency unchanged: $frequencies[$frequency]\n");
  }
} elsif ($deltaR < 0) {        # Delta R increment was negative
  if($frequency != $f_min) {  # Calculate threshold
    $threshold = $frequencies[$frequency+1]*$threshold_down;
  } else {
    $threshold = $frequencies[$f_min]*$threshold_down;
  }
  print(OUTPUT "Decrement threshold: $threshold\n");
  # Check if resource has reached threshold
  if($resource <= $threshold) {
    # Check if frequency is already minimum
    if($frequency != $f_min) {
      print(OUTPUT "Frequency $frequencies[$frequency] changed to ");
      $frequency++;            # Adjust frequency level downwards
      print(OUTPUT "$frequencies[$frequency]\n");
      system("cpufreq-set", "-f", "$frequencies[$frequency]");
    } else {
      print(OUTPUT "Already at lowest available frequency\n");
    }
  } else {
    print(OUTPUT "Frequency unchanged: $frequencies[$frequency]\n");
  }
} else {
  print(OUTPUT "Resource unchanged\n");
}

# Store frequency set (or kept) in this iteration
my $freqtemp = $frequencies[$frequency]/1000000;
```

```
    push(@setfreqs, "$freqtemp\n");

    # Make loop wait a little before next iteration
    sleep $sleep;
}
close(OUTPUT);

# END
```

Appendix B

Additional graphs

Figures B.1 and B.2 show low and full constant load.

Figures B.3 and B.4 show low and full constant load with added noise.

Figure B.5 shows sinusoidal load with noise.

Figure B.6 shows variable load with noise.

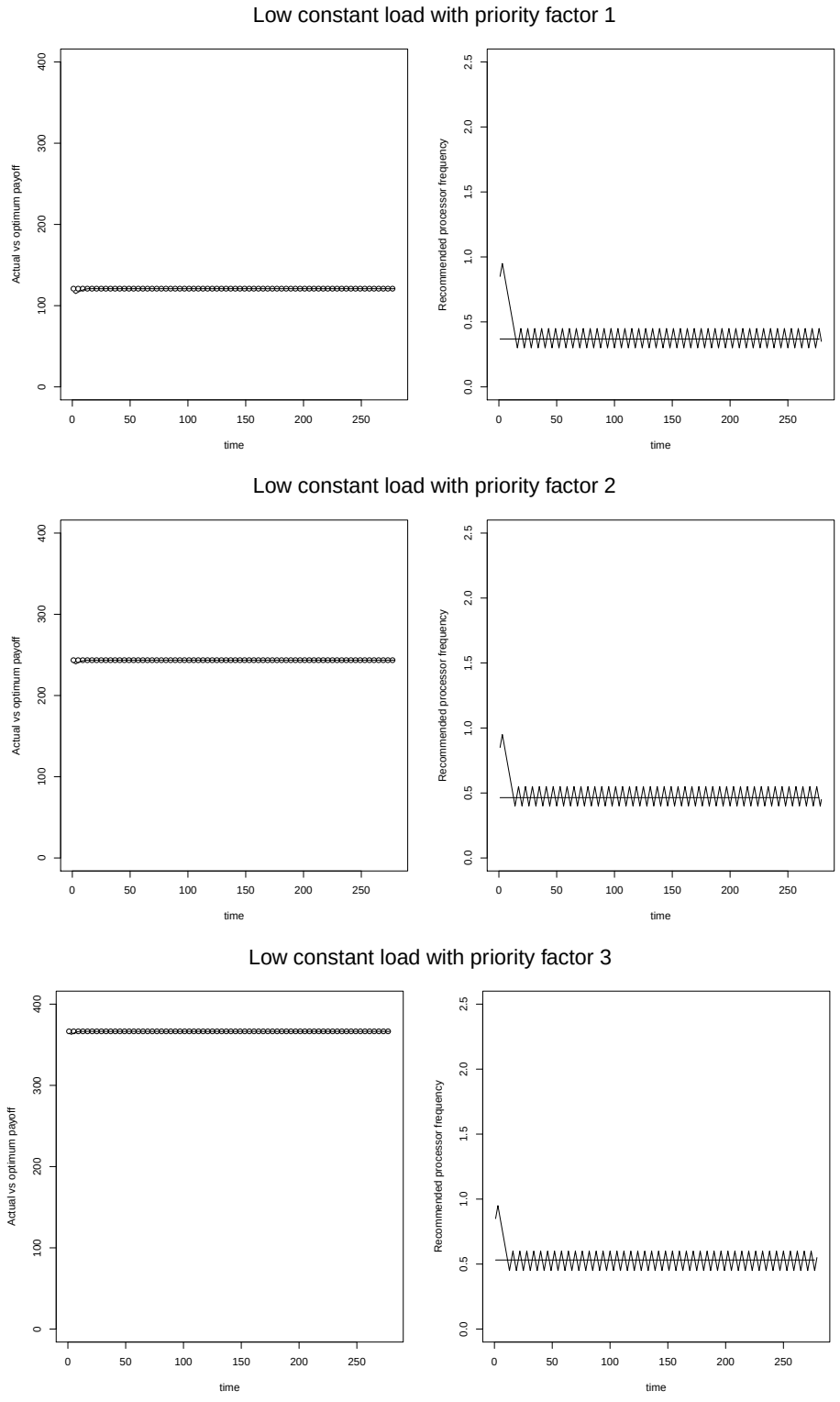

Figure B.1: Actual vs. optimal payoff and optimal vs. recommended resource values with low constant load (1%) and priority factors 1, 2 and 3. Optimum is shown with circles in the left graphs and with a straight line in the right graphs.

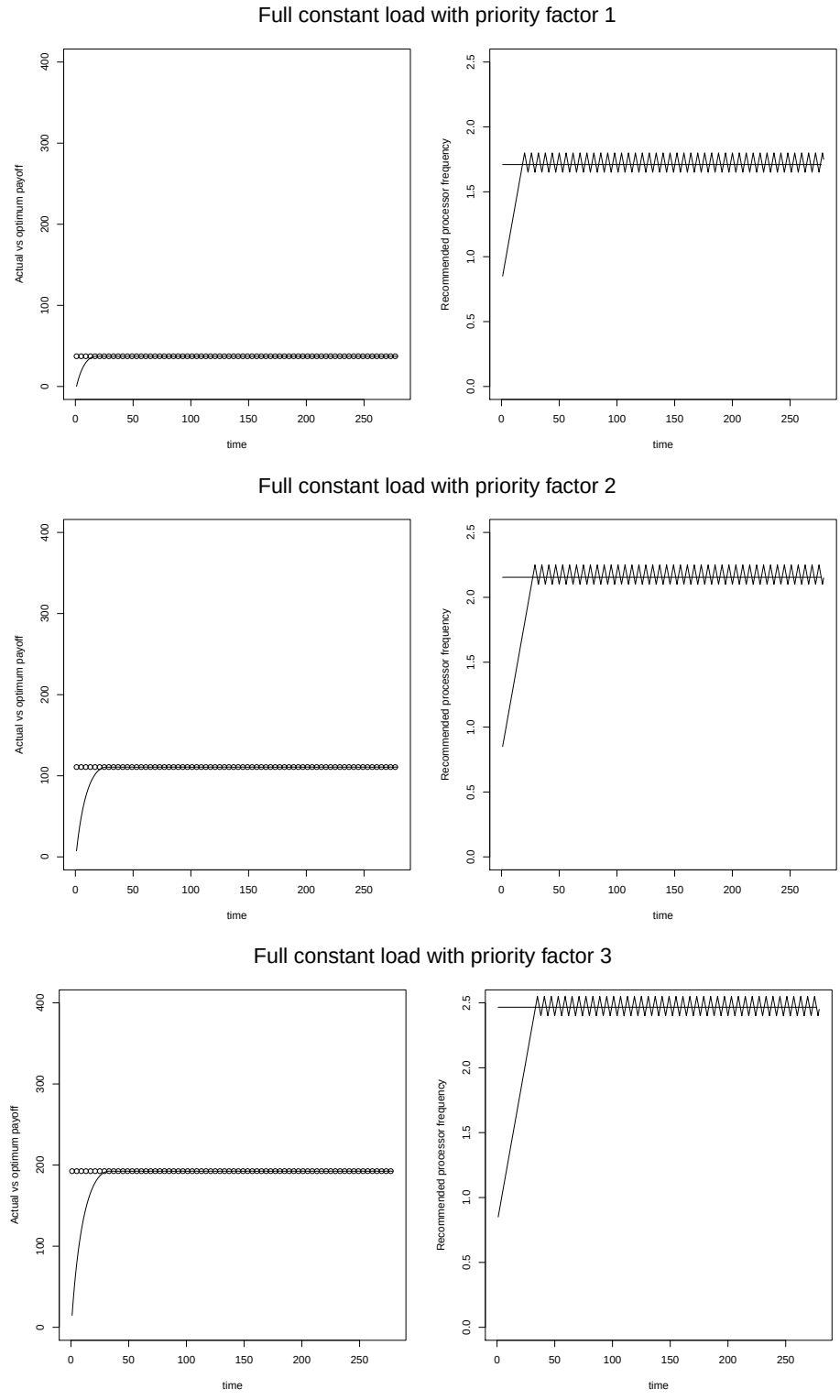

Figure B.2: Actual vs. optimal payoff and optimal vs. recommended resource values with full constant load (100%) and priority factors 1, 2 and 3. Optimum is shown with circles in the left graphs and with a straight line in the right graphs.

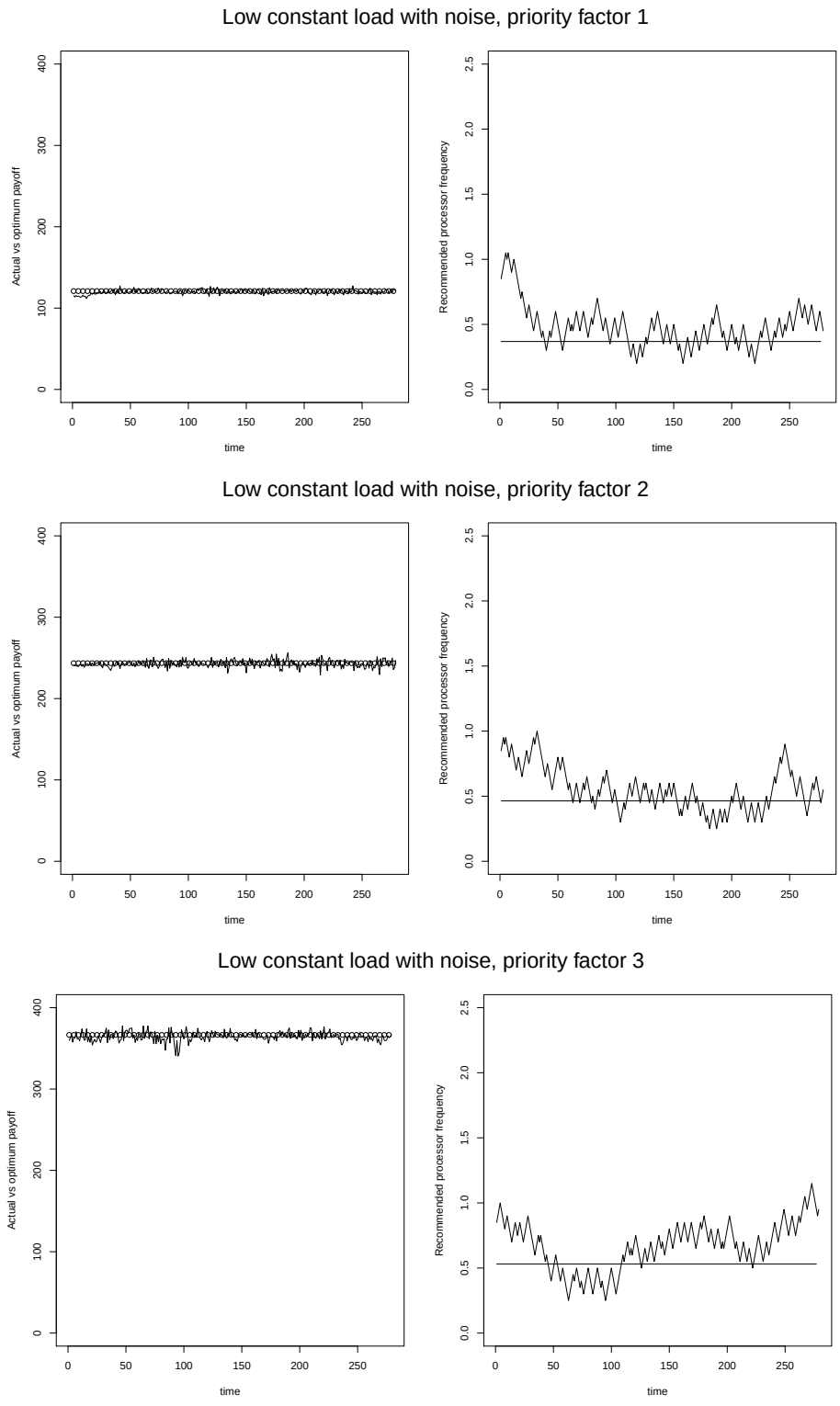

Figure B.3: Actual vs. optimal payoff and optimal vs. recommended resource values with low constant load (1%), added noise and priority factors 1, 2 and 3. Optimum is shown with circles in the left graphs and with a straight line in the right graphs.

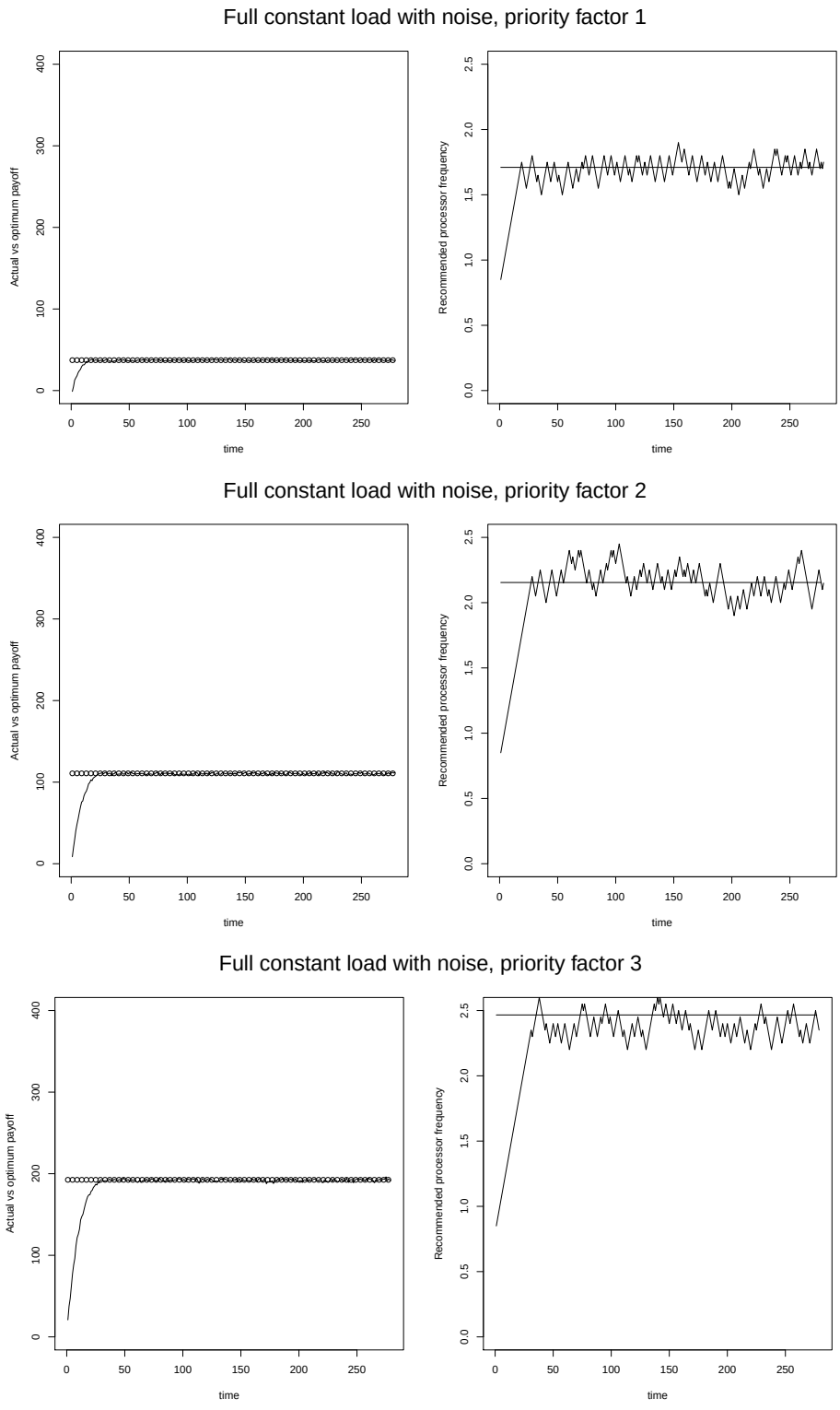

Figure B.4: Actual vs. optimal payoff and optimal vs. recommended resource values with full constant load (100%), added noise and priority factors 1, 2 and 3. Optimum is shown with circles in the left graphs and with a straight line in the right graphs.

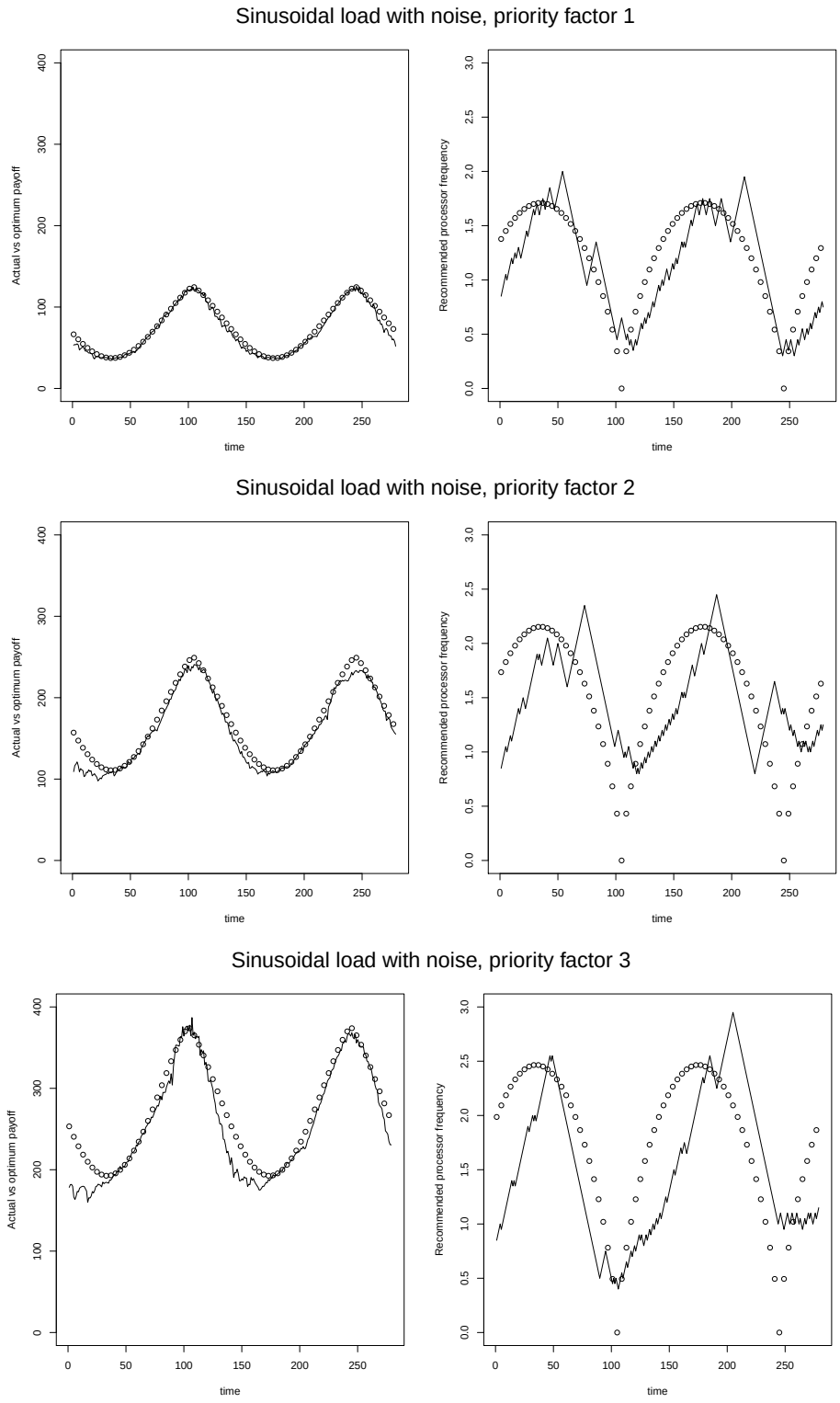

Figure B.5: Actual vs. optimal payoff and optimal vs. recommended resource values with sinusoidal load, added noise and priority factors 1, 2 and 3. Optimum is shown with circles.

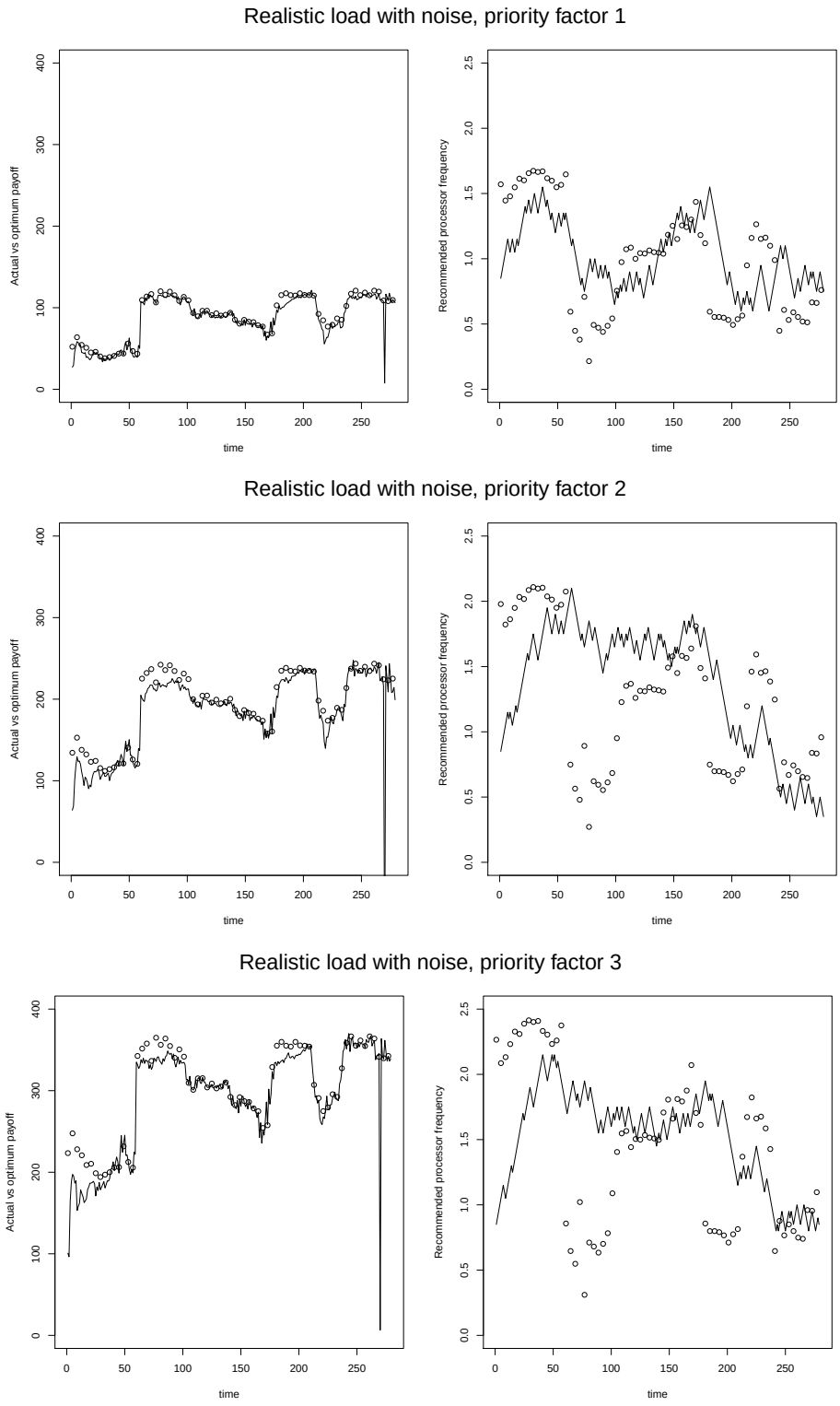

Figure B.6: Actual vs. optimal payoff and optimal vs. recommended resource values with variable load, added noise and priority factors 1, 2 and 3. Optimum is shown with circles.